A gas kinetic scheme approach for modeling and simulation of fire on massively parallel hardware

Von der
Fakultät Architektur, Bauingenieurwesen und Umweltwissenschaften
der Technischen Universität Carolo-Wilhelmina
zu Braunschweig

zur Erlangung des Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertation

von
Stephan Lenz
geboren am 09. November 1989
aus Höxter

Eingereicht am:	19. November 2019
Disputation am:	3. März 2020
Berichterstatter:	Prof. Dr.-Ing. habil. Manfred Krafczyk
	Prof. Dr. rer. nat. Martin Geier

2020

Acknowledgments

First of all, I want to thank my doctoral advisor Prof. Manfred Krafczyk. Thank you for giving me the opportunity to work on this interesting project. And while you gave me the freedom to work on this project independently and in my own time, you managed to keep me on track towards the point where I stand now.

Second, but not a bit less, I want to thank my office roommate Prof. Martin Geier. You always had time for me, looking at my results, discussing theory and showing me how much more there is to know in the world. I am honestly sorry, Martin, for the all the drums I played on my desk and all the catchy tunes I just had to sing along.

This dissertation would not have come into being without GRK 2075. I want to thank the whole board for, first, proposing this project and, second, accepting me as part of it. The GRK was a great opportunity to meet other Ph.D. students from outside the institute and learn a lot about civil engineering. GRK also gave me the possibility to present my work and get valuable feedback during our biannual workshops and at conferences in Europe and North-America.

Having colleagues you can talk to whenever you do not know how to proceed with your work (or when you just have to stretch your legs) makes research possible in the first place. I am very grateful to Martin S., Konstantin, Hussein, Hesam, Jan and Sören for all the things they have helped me with during this project. Writing a dissertation is never research alone, though. I want to thank our IT staff Fritz, Michael and Helmut for helping me out with pretty much anything from setting up my PC, getting my code to run on the cluster, fixing broken hardware or just accompanying us to lunch. A very special thanks belongs to our secretary Anne. She never failed to support me in any matter (administrational or other), including not-so-straight-forward travel expanse bills. To everybody at iRMB I have to say: Thank you very much, for providing such a nice work environment.

Finally, I want to thank my family for always supporting me on my way through life and for believing in me. This thanks goes especially to my girlfriend, who had to endure all my ups and downs during the past years. She even found time to read this whole thesis and point out several minor mistakes and hundreds of misplaced commas, while she had important things going on in her life as well. Thank you so much for being there for me!

Abstract

This work presents a simulation approach based on a Gas Kinetic Scheme (GKS) for the simulation of fire that is implemented on massively parallel hardware in terms of Graphics Processing Units (GPU) in the framework of General Purpose computing on Graphics Processing Units (GPGPU).

Gas kinetic schemes belong to the class of kinetic methods because their governing equation is the mesoscopic Boltzmann equation, rather than the macroscopic Navier-Stokes equations. Formally, kinetic methods have the advantage of a linear advection term which simplifies discretization. GKS inherently contains the full energy equation which is required for compressible flows. GKS provides a flux formulation derived from kinetic theory and is usually implemented as a finite volume method on cell-centered grids. In this work, we consider an implementation on nested Cartesian grids. To that end, a coupling algorithm for uniform grids with varying resolution was developed and is presented in this work. The limitation to local uniform Cartesian grids allows an efficient implementation on GPUs, which belong to the class of many core processors, i.e. massively parallel hardware. Multi-GPU support is also implemented and efficiency is enhanced by communication hiding.

The fluid solver is validated for several two- and three-dimensional test cases including natural convection, turbulent natural convection and turbulent decay. It is subsequently applied to a study of boundary layer stability of natural convection in a cavity with differentially heated walls and large temperature differences.

The fluid solver is further augmented by a simple combustion model for non-premixed flames. It is validated by comparison to experimental data for two different fire plumes. The results are further compared to the industry standard for fire simulation, i.e. the Fire Dynamics Simulator (FDS). While the accuracy of GKS appears slightly reduced as compared to FDS, a substantial speedup in terms of time to solution is found. Finally, GKS is applied to the simulation of a compartment fire.

This work shows that the GKS has a large potential for efficient high performance fire simulations.

Zusammenfassung

Diese Arbeit präsentiert einen Simulationsansatz basierend auf einer gaskinetischen Methode (eng. Gas Kinetic Scheme, GKS) zur Simulation von Bränden, welcher für massiv parallel Hardware im Sinne von Grafikprozessoren (eng. Graphics Processing Units, GPUs) implementiert wurde.

GKS gehört zur Klasse der kinetischen Methoden, die nicht die makroskopischen Navier-Stokes Gleichungen, sondern die mesoskopische Boltzmann Gleichung lösen. Formal haben kinetische Methoden den Vorteil, dass der Advektionsterms linear ist. Dies vereinfacht die Diskretisierung. In GKS ist die vollständige Energiegleichung, die zur Lösung kompressibler Strömungen benötigt wird, enthalten. GKS formuliert den Fluss von Erhaltungsgrößen basierend auf der gaskinetischen Theorie und wird meistens im Rahmen der Finiten Volumen Methode umgesetzt. In dieser Arbeit betrachten wir eine Implementierung auf gleichmäßigen Kartesischen Gittern. Dazu wurde ein Kopplungsalgorithmus für die Kombination von Gittern unterschiedlicher Auflösung entwickelt. Die Einschränkung auf lokal gleichmäßige Gitter erlaubt eine effiziente Implementierung auf GPUs, welche zur Klasse der massiv parallelen Hardware gehören. Des Weiteren umfasst die Implementierung eine Unterstützung für Multi-GPU mit versteckter Kommunikation.

Der Strömungslöser ist für zwei und dreidimensionale Testfälle validiert. Dabei reichen die Tests von natürlicher Konvektion über turbulente Konvektion bis hin zu turbulentem Zerfall. Anschließend wird der Löser genutzt um die Grenzschichtstabilität in natürlicher Konvektion bei großen Temperaturunterschieden zu untersuchen.

Darüber hinaus umfasst der Löser ein einfaches Verbrennungsmodell für Diffusionsflammen. Dieses wird durch Vergleich mit experimentellen Feuern validiert. Außerdem werden die Ergebnisse mit dem gängigen Brandsimulationsprogramm FDS (eng. Fire Dynamics Simulator) verglichen. Die Qualität der Ergebnisse ist dabei vergleichbar, allerdings ist der in dieser Arbeit entwickelte Löser deutlich schneller. Anschließend wird das GKS noch für die Simulation eines Raumbrandes angewendet.

Diese Arbeit zeigt, dass GKS ein großes Potential für die Hochleistungssimulation von Feuer hat.

Contents

1 Introduction

The taming of fire is one of the most important achievements of mankind on its way to modern humanity. But, albeit having tamed fire for a million years [1], it still has a great potential to harm humans and their environment. Wildfires can easily destroy large forested areas, endangering the lives and health of people, destroy infrastructure and harm the climate. Building fires posses an even greater danger to the lives of humans, due to small compartments and narrow escape paths. In Germany about 300 to 400 people die due to exposition to smoke, fire and flames every year [2, 3]. Just recently the fire in the Grenfell tower in London, UK, caused the death of 72 people [4]. This incident went through the news world wide and caused safety reevaluations of many residential builds around the world. Hence, fire safety is an important design criterion for buildings. The investigation of fire safety comprises multiple aspects, such as structural safety, fire spread, fire suppression, smoke dispersion and evacuation.

Experimental investigations of these aspects are difficult, because buildings are often only built once. For the safety of a car, one can build ten cars, crash them and deduce safety aspects from the results. For a building that is only built once, it is obviously not feasible to build it first, then perform fire safety experiments and finally build it again for usage. Hence, fire safety is mostly based on small scale experiments of exemplary setups.

With substantial improvements in the accuracy of numerical methods and increasing computational power over the last century, more and more engineering experiments are complemented by numerical experiments. In simulations, the degree of detail can easily be adjusted, such that even the behavior of large systems can be computed. Hence, it is possible to predict the behavior of a technical system even before it is built. For fire safety investigations, this allows to analyze the specific design of a building.

The simulation of fire is not a trivial task. It is a multi physics and multi scale problem. The main component is a fluid solver for thermal compressible fluids with chemical reactions for the combustion. Further, effects as pyrolysis, phase change, radiative heat transfer, fire suppression systems and so on need to be considered. The spatial scales of fire range from tens and hundreds of meters (i.e. the size of buildings) to the sub-millimeter scales (e.g. in the flame thickness). Numerical codes that incorporate all these effects are nowadays available, e.g. [5, 6]. However, these codes do not scale well with recent improvements in high performance computing, which are mostly due to massive parallelization.

The aim of this work is to investigate a non-classical flow solver based on a Gas Kinetic Scheme (GKS) [7] for its applicability in the simulation of fire. This work

mainly focuses on the flow solver and its efficient parallel implementation. With respect to reaction modeling, a simple model is considered.

The scientific field of fluid dynamics deals with the motion of fluids. The governing equations of fluid motion that we can observe in every day life are the Navier-Stokes equations, that imply conservation equations for mass, momentum and energy. A large number of numerical methods and schemes has been developed and used over the last decades. The application of these methods is the field of Computational Fluid Dynamics (CFD), which comprises topics from physics, mathematics, engineering and computer sciences. In classical CFD, the Navier-Stokes equations are discretized and solved numerically on Eulerian grids. Some non-classical methods use discrete Lagrangian particles to approximate the fluid, e.g. Smoothed Particle Hydrodynamics (SPH) [8]. Others belong to the class of kinetic methods for CFD. They are derived from gas kinetic theory, which is a description of gases on the mesoscopic scale, i.e. a statistical description of particle motion. Often kinetic methods are employed for non-classical fluid dynamics, such as small scale and rarefied flows. The governing equation on the mesoscopic scale is the Boltzmann equation, which describes the evolution of particles in a statistical fashion. The Navier-Stokes equations are a limit of the Boltzmann equation if the system is much larger than the mean free path of the particles. This property is used in the construction of kinetic schemes for classical fluid dynamics. While these schemes are derived from the Boltzmann equation, they usually solve problems formulated in terms of the Navier-Stokes equations. The most popular of such methods is the Lattice Boltzmann Method (LBM), which is widely used in science and engineering, mostly for incompressible flows [9, 10, 11, 12]. The simulation of compressible flows becomes inefficient due to large stencils required for recovering the energy equation, e.g. [13]. In GKS, the energy equation is an inherent part of the scheme, such that GKS is directly applicable to compressible flows. Kinetic methods are usually explicit and local because the propagation of information is connected to particle transport and interaction, which happens locally. This distinguishes them from many classical CFD methods. Due to this property, kinetic methods, including GKS, can easily be implemented on massively parallel hardware, as it is available today.

GKS is usually implemented as a finite volume method, where mass, momentum and energy are tracked. The fluxes are derived as moments of a mesoscopic particle distribution function, which is constructed from the tracked data. It was originally introduced for supersonic flows with shock capturing [7], but can be simplified for thermal compressible flows [14, 15]. As a finite volume method it can in general be implemented on unstructured grids with arbitrary cell shapes [16, 17, 18, 19]. This work features an implementation on nested uniform Cartesian grids for the sake of simplifying automated grid generation and enhancing computational efficiency. An algorithm for the coupling of octree based grids is proposed and tested. The topic of automated grid generation is also discussed in this work and a grid generator for both LBM and GKS is presented. It features octree-based refinement, complex geometries based on a cut-cell approach, thin walls and a simple domain decomposition for distributed computing. Not all of these features are applicable to GKS. In terms

of complex geometries, a novel morph-cell approach for the finite volume method is presented.

Massively parallel hardware can be subdivided into, first, large cluster computers with hundreds and thousands of independent computers and, second, integrated many core processors with thousands of cores in a single processor. The latter approach is found for instance in Graphics Processing Units (GPUs). The simulation code that was developed in the course of this work is VirtualFluidsGKS, which is part of the VirtualFluids software package [20]. VirtualFluids currently comprises three kinetic solvers for massively parallel hardware. VirtualFluidsGKS implements GKS for the concurrent usage of multiple GPUs.

In order to evaluate the applicability of GKS for fire simulation, VirtualFluidsGKS is validated with several tests, ranging from two- and three-dimensional natural convection, over three-dimensional turbulence to the simulation of fire plumes, where VirtualFluidsGKS results are compared against experiments. Further, the results are compared to state-of-the-art fire simulation software, in order to evaluate computational efficiency.

In this thesis Chapter 2 is devoted to GKS. Therein, first the basics of gas kinetic theory are introduced and the connection between the Boltzmann equation and the Navier-Stokes equations is shown. A literature review of GKS variants follows. The specific variant considered in this work is derived in detail and its realization in the finite volume method is described. A novel approach for coupling grids of different resolution is proposed and tested. In Chapter 3 fire modeling is reviewed and the combustion model used in this work is discussed. Then, after introducing the theory, the Multi-GPU implementation of VirtualFluidsGKS is described in Chapter 4 and computational performance is investigated. The topic of grid generation is discussed in Chapter 5. This includes the introduction and evaluation of the morph-cell algorithm. The validity and applicability of VirtualFluidsGKS for flow and fire simulations is investigated in Chapter 6. Finally, this work is summarized and concluded in Chapter 7, where also an outlook of future work is given.

2 The gas kinetic scheme

2.1 On gas kinetic theory

The way we describe matter depends on the scale of observation. In daily life, matter appears to us as a continuum. In reality matter is composed of discrete particles, i.e. atoms and molecules. The particles on this microscopic scale obey relatively simple equations of motion. The complex behavior of matter is originated in the interplay of zillions of such particles. On the macroscopic scale, the evolution of matter is described by partial differential equations that model the phenomena observed in daily life. These partial differential equations are usually based on first principles of physics, such as conservation of mass, momentum and energy, and complemented with constitutive laws, such as diffusive processes, e.g. viscosity and heat transfer.

The mesoscopic scale resides in between the microscopic and macroscopic scale. It considers the discrete particles only in a statistical way, where information of distinguishable particles is lost. The evolution of the particle statistics is modeled by partial differential equations. For gases, the mesoscopic scale is described by the gas kinetic theory.

The following sections introduce the basic concepts of gas kinetic theory that are required for the understanding of gas kinetic schemes for fluid mechanics.

2.1.1 Basic relations

Due to the distinction between microscopic and macroscopic scales, some non-standard notations are required. The discrete particles on the microscopic scale move in three dimensional space with velocity $\vec{u} = (u, v, w)^T$. Averaging over large ensembles of particles yields the velocity of the continuum $\vec{U} = (U, V, W)^T$. The connection between the velocity on the scales is

$$\vec{U} = \frac{1}{N} \sum_{i=1}^{N} \vec{u}_i, \tag{2.1}$$

where N is the ensemble size. On the microscopic scale, the particles are in constant motion and momentum is constantly exchanged between particles by collision. The mean distance a particle travels between collisions is called mean free path l. The

ratio of mean free path and a characteristic length scale of the system L define the dimensionless Knudsen number [21, Chapter 1.4.1]

$$Kn = \frac{l}{L}. \tag{2.2}$$

High Knudsen numbers occur for dilute gases, such as in the upper atmosphere, or in very small systems, e.g. micro mechanical system. High Knudsen numbers imply that the effects on the microscopic scale are not negligible and collisions are rare. The limit of infinite Knudsen number describes collisionless particle motion. Low Knudsen numbers describe gases, where a continuum description is valid, i.e. the system is much larger than the mean free path. Collisions happen frequently in this regime. This regime is targeted in this work.

The microscopic particle motion is closely related to its internal energy

$$\tilde{e}_P = \frac{1}{2}\tilde{m}((u-U)^2 + (v-V)^2 + (w-W)^2 + \xi_1^2 + \cdots + \xi_K^2), \tag{2.3}$$

where ξ denotes a velocity along an internal degree of freedom in the molecule and $\tilde{m}$ is the molecular mass. Atoms do not have internal degrees of freedom. The mass of an atom is concentrated in the nucleus, which is of negligible volume, such that the nucleus has no rotational inertia, which could store energy. Molecules, such as the main components of air N_2 and O_2, have a distributed mass, and can hence store energy in the rotation of the molecule. Further it is possible that the atoms in the molecule vibrate and, hence, change their relative positions. These vibrations can also store energy, but they are usually frozen at moderate temperatures. The number of internal degrees of freedom is denoted by K, such that a molecule would have $K+3$ total degrees of freedom. The equipartition theorem (cf. [22, Chapter 9.5]) states that in equilibrium the internal energy of a molecule is distributed evenly over the degrees of freedom. The energy per degree of freedom can be computed by Boltzmann constant k_B and absolute Temperature T. The internal energy of a molecule in terms of temperature is [21, Chapter 2.1.2]

$$\tilde{e}_P = \frac{K+3}{2}k_B T. \tag{2.4}$$

It is usually not feasible to look at single particles but ensembles of particles. The amount of substance n, i.e. the number of particles, is measured in multiples of the Avogadro constant N_A, which historically was defined as the number of carbon atoms (with a molecular mass of $\tilde{m}_C = 12\ u$) in 12 g pure carbon. In 2019, N_A was redefined to a fixed value in close relation to the previous definition [23].

The energy of an ensemble can then be computed as

$$\tilde{e} = nN_A\tilde{e}_P = \frac{K+3}{2}nN_A k_B T = \frac{K+3}{2}nR_u T, \tag{2.5}$$

where $R_u = N_A k_B$ is the universal gas constant. Introducing the molar mass as $M = \tilde{m}N_A$ and the specific gas constant as $R = R_u/M = k_B/\tilde{m}$ yields the internal

energy in the form used in this work as

$$e = \frac{\tilde{e}}{nM} = \frac{K+3}{2}RT, \tag{2.6}$$

where e denotes the internal energy per unit mass.

The number of internal degrees of freedom has a one-to-one relation to the more conventional ratio of specific heats

$$\gamma = \frac{K+5}{K+3} \quad \Leftrightarrow \quad K = \frac{5-3\gamma}{\gamma-1}. \tag{2.7}$$

The (mass specific) heat capacity at constant volume can be read directly from Eq. (2.6) as the proportionality factor between temperature and energy, i.e.

$$c_v = \frac{K+3}{2}R \tag{2.8}$$

As described above, the number of internal degrees of freedom K is not constant over temperature and, hence, neither is the heat capacity. For the sake of linearity in the energy-temperature transformation this is neglected in this work. Further relations regarding heat capacities are found in [22, Chapter 9.5]:

$$\gamma = \frac{c_p}{c_v} \ , \qquad c_p - c_v = R \ , \qquad c_p = \frac{K+5}{2}R \quad \Leftrightarrow \quad K = 2\frac{c_p}{R} - 5 \tag{2.9}$$

In this work we assume the heat capacity to be independent of the temperature. This is physically not correct. The number of degrees of freedom depends on temperature, because some degrees of freedom are frozen at low temperatures and get activated at larger temperatures. This is especially relevant for complex molecules. The relation between temperature and energy is then

$$e = \int_0^T c_v(T)dT. \tag{2.10}$$

The data for $c_v(T)$ can be obtained from the NIST-JANAF Thermochemical Tables [24]. The forward computation of energy e is possible by numerical integration of the temperature dependent heat capacity. Extraction of temperature requires iterative solution of the integral equation. Such a procedure has severe implications on computational efficiency and was, hence, neglected in this work after some initial tests. The implication of the assumption of constant heat capacity on the results should be low, because air and combustion products are usually simple molecules, where the temperature dependence is small.

It is possible to derive the most probable particle speed $|\vec{u}|_p = \sqrt{2RT}$ from kinetic theory [21, Chaper 4.1.3]. Later in this work the squared inverse most probable particle speed

$$\lambda = \frac{1}{2RT} \tag{2.11}$$

will be used instead of the temperature. Hence, all equations become independent of the specific gas constant R.

Gas kinetic theory also predicts a value for the speed with which information travels through the gas, i.e. the speed of sound [21, Chapter 2.1.2]. It is computed as

$$c_s = \sqrt{\gamma RT}. \tag{2.12}$$

Finally, the pressure p can be compute from the ideal gas law [25] as

$$p = \rho RT = \frac{\rho}{2\lambda}. \tag{2.13}$$

Therein, ρ is the density, i.e. the volume specific mass, such that $\rho = m/V = nM/V$.

2.1.2 The mesoscopic scale

The prior section briefly introduced the connection between microscopic and macroscopic scale. The mesoscopic scale resides in between the two scales. Instead of considering individual particles, it observes statistics of ensembles of particles, such that the information about individual particles is lost. The macroscopic scale is constructed from low order statistics of the particle properties, while the mesoscopic scale keeps all statistical moments. Hence, the state is modeled by probability density functions instead of a fixed state. The state variable on the mesoscopic scale is the particle distribution function in phase-space

$$\tilde{f} = \tilde{f}(\vec{x}, \vec{u}, \xi_1, \ldots, \xi_K, t). \tag{2.14}$$

It encodes all macroscopic quantities in a single function that lives in a high-dimensional (more accurately $K + 6$ dimensional plus time) space [26, Chapter 2.2]. This makes analytic solutions on the mesoscopic scale difficult.

An infinitesimal volume element in phase space is $dxdydzd\Xi$ with the velocity space element $d\Xi = dudvdwd\xi_1 \ldots d\xi_K$. Taking the product $\tilde{f}(\vec{x}, \vec{u}, \xi_1, \ldots, \xi_K, t)dxdydzd\Xi$ gives the probability to find a particle in an infinitesimal volume element in phase space. Integrating this distribution function over the whole ensemble, i.e. the whole phase space yields one. Integrating over velocity space only, gives the number density at a certain point in space.

For practical purposes the particle density function is scaled by the ensemble mass m [26, Chapter 2.2], such that

$$f = m\tilde{f}. \tag{2.15}$$

The connection from the mesoscopic scale to the macroscopic scale is given in terms of velocity space moments of the scaled particle distribution function f. Since the integral of $\tilde{f}$ over the whole phase space, must yield one, the same integral of the

scaled particle distribution function f must yield the ensemble mass m. Hence, the density can be computed from [26, Chapter 2.2]

$$\rho(\vec{x},t) = \int_{\Xi} f(\vec{x},\vec{u},\xi_1,\ldots,\xi_K,t)d\Xi. \tag{2.16}$$

Statistically, the first moment of a distribution is the mean. Hence, the first order moments of the particle distribution function $\tilde{f}$ are the macroscopic velocity components. The first order moments of the scaled distribution function f yield the momentum components [27, Chapter 2.1]

$$\rho\vec{U} = \int_{\Xi} \vec{u} f d\Xi. \tag{2.17}$$

Note that the mean vanishes for all internal degrees of freedom, i.e.

$$0 = \int_{\Xi} \xi_i f d\Xi \tag{2.18}$$

for all $i = 1,\ldots,K$.

The internal energy is introduced in Eq. (2.3) as a kinetic energy on the microscopic scale. The same procedure can be performed on the mesoscopic scale leading to the macroscopic internal energy [27, Chapter 2.1]

$$\rho e = \int_{\Xi} \frac{1}{2}\left((\vec{u}-\vec{U})^2 + \xi^2\right) f d\Xi, \tag{2.19}$$

where $\xi^2 = \xi_1^2 + \cdots + \xi_K^2$. Expanding the integral yields

$$\rho e = \underbrace{\int_{\Xi} \frac{1}{2}\left(\vec{u}^2 + \xi^2\right) f d\Xi}_{\rho E} - \frac{1}{2}\vec{U}^2 \int_{\Xi} f d\Xi = \rho E - \frac{\rho}{2}\vec{U}^2, \tag{2.20}$$

where $E = e + \vec{U}^2/2$ is considered to be the total mass specific energy consisting of internal energy and kinetic energy on the macroscopic scale. Inserting Eq. (2.6) yields the transformation between total energy and temperature as

$$E = \frac{K+3}{2}RT + \frac{1}{2}\vec{U}^2 = \frac{K+3}{4\lambda} + \frac{1}{2}\vec{U}^2. \tag{2.21}$$

The state on the macroscopic scale can be given in terms of primitive variables $\underline{Z} = (\rho, \vec{U}, \lambda)^T$ as well as in terms of conserved variables $\underline{W} = (\rho, \rho\vec{U}, \rho E)^T$. The momentum velocity transformation is trivial. The transformation between energy and temperature is given in Eq. (2.21).

The transformation from the scaled particle distribution function f to the conserved variables is given by the conserved moments [7]

$$\underline{W} = \int_{\Xi} \underline{\psi} f d\Xi, \tag{2.22}$$

where $\psi = (1, \vec{u}, 1/2(\vec{u}^2 + \xi^2))$ are the collision invariants, see [28, Chapter 4.1.1]. Collisions will be introduced below. The backward transformation is not directly possible. The particle distribution function f contains more information than the macroscopic quantities by taking into account an infinite number of statistical moments. This is obvious, since the conserved quantities are constructed from low order moments of f.

Up to this point only the state at a distinct point in time was considered. Nevertheless, the particles are in constant motion. Hence, an instantaneous state may not remain constant over time. In fact, the particles act in two ways. First, they move through space with their particle velocity. This process is called advection. Second, they exchange momentum by various forces. The main contribution of momentum transfer between particles stems from close distance repulsion forces that appear, when particles come very close to each other. This effect is termed collision, even though the concept of two solid objects "colliding" is not accurate on the microscopic scale. Rather, a growing repulsion force prevents the "collision" and forces the particle away from each other. This process is energy conserving and can be modeled as an elastic impact.

The constant collision and advection of particles prevent the system from reaching a steady state at the microscopic scale. The question is whether a steady state can be obtained on the mesoscopic scale. The answer to this question lies in the thermodynamical equilibrium. From a macroscopic perspective, a gas is in thermodynamical equilibrium, if the macroscopic state has no gradients [21, Chapter 4]. Hence, thermodynamical equilibrium is a steady state. Further, non-equilibrium states can be steady, e.g. steady heat transfer.

From statistical mechanics, the equilibrium state can be described as the mesoscopic distribution that has the most possible microscopic states [25, Chapter 1.3]. Fortunately, it is possible to compute the particle distribution for an equilibrium state as

$$f^{eq} = \rho \left(\frac{\lambda}{\pi}\right)^{\frac{K+3}{2}} \exp\left(-\lambda((\vec{u} - \vec{U})^2 + \xi^2)\right) \tag{2.23}$$

for a given macroscopic state. The equilibrium distribution is called Maxwellian distribution [29]. It resembles a Gaussian distribution centered around the macroscopic velocity. The symmetry of the Gaussian distribution implies that in the moving frame of reference, the same number of particles with a given velocity travel in two opposing directions. A state that is not represented by a Maxwell distribution is said to be in non-equilibrium. The non-equilibrium distribution may be arbitrary.

A distribution can be split into equilibrium and non-equilibrium parts

$$f = f^{eq} + f^{neq}, \tag{2.24}$$

where only the equilibrium part f^{eq} contains the information about the macroscopic state and, hence, the moments of the non-equilibrium part f^{neq} with respect to collision invariants disappear, i.e.

$$0 = \int_{\Xi} \underline{\psi} f^{neq} d\Xi. \tag{2.25}$$

A helpful feature of the Maxwellian equilibrium distribution f^{eq} is the fact that its moments can be evaluated analytically. A useful notation for moments of the non-scaled distribution function $\tilde{f}$ is used by Xu [26]. The notation for the first moments with respects to the velocity component u is

$$\langle 1 \rangle = \int_{\Xi} \tilde{f}^{eq} d\Xi, \quad \langle u \rangle = \int_{\Xi} u \tilde{f}^{eq} d\Xi, \quad \left\langle u^2 \right\rangle = \int_{\Xi} u^2 \tilde{f}^{eq} d\Xi, \quad \ldots. \tag{2.26}$$

The same notation is used for the other velocity components, including the velocities of internal degrees of freedom, e.g. $\langle v \rangle$, $\langle w^2 \rangle$, $\langle \xi^4 \rangle$, where the internal degrees of freedom are collected in the squared speed $\xi^2 = \xi_1^2 + \cdots + \xi_K^2$. Mixed moments are denoted by

$$\langle uv \rangle = \int_{\Xi} uv \tilde{f}^{eq} d\Xi. \tag{2.27}$$

The equilibrium distribution function can be factorized as $\tilde{f}^{eq} = \tilde{f}_u^{eq} \tilde{f}_v^{eq} \tilde{f}_w^{eq}$ due to the independence of the equilibrium distribution over the velocity components. In the same manner the moments can also be factorized as

$$\langle uv \rangle = \langle u \rangle \langle v \rangle . \tag{2.28}$$

The moments follow the recursive formulation

$$\left\langle u^0 \right\rangle = 1, \quad \langle u \rangle = U, \quad \left\langle u^n \right\rangle = U \left\langle u^{n-1} \right\rangle + \frac{n-1}{2\lambda} \left\langle u^{n-2} \right\rangle \tag{2.29}$$

and similar for the other velocity components v and w [26, Appendix C]. According to Eq. (2.18), the first moments with respect to the internal degrees of freedom vanish. Hence, all uneven moments vanish, i.e. $\langle \xi^{2n+1} \rangle = 0$. The moments $\langle \xi^{2n} \rangle$ account for the sum of the moments with respect to K internal degrees of freedom, such that the moments are multiplied by K. The resulting recursive formulation is

$$\left\langle \xi^0 \right\rangle = 1, \quad \left\langle u^{2n} \right\rangle = \frac{K + 2(n-1)}{2\lambda} \left\langle \xi^{2(n-1)} \right\rangle . \tag{2.30}$$

2.1.3 The BGK-Boltzmann equation

After introducing the mesoscopic state description, this section deals with the temporal evolution of the mesoscopic state, i.e. the scaled particle distribution function. For the sake of simplicity, the attribute "scaled" will be omitted from here on.

The particle distribution function is a continuous function in space and time. Hence, its temporal evolution can be modeled by a general transport equation

$$\frac{\partial f}{\partial t} + \vec{u} \cdot \nabla f = F, \tag{2.31}$$

where the term F collects all the forces acting on the particles. Neglecting forces, the transport equation for f states that the probability of a particle is advected with the corresponding velocity of the particle.

For the force term F two contributions are considered. Forces coming from a gravitational field $\vec{g}$ are modeled by $\vec{g} \cdot \nabla_{\vec{u}} f$. Observing the similarity to the advective part, the gravitational forcing can be imagined as a constant advection in velocity space. The second set of forces are the collision forces, denoted by $\Omega(f, f)$. For several reasons the collision term is very complicated. Particles of any two differing velocities can collide, hence the collision term needs a double integration over the velocity space. Furthermore, the velocity direction after the collision depends on the velocity directions before collision. The full Boltzmann equation reads [30]

$$\frac{\partial f}{\partial t} + \vec{u} \cdot \nabla f + \vec{g} \cdot \nabla_{\vec{u}} f = \Omega(f, f). \tag{2.32}$$

For the full form of the collision term and detailed explanation see e.g. [31, Section 2]. The collision must conserve mass, momentum and energy, i.e.

$$0 = \int_{\Xi} \underline{\psi} \Omega(f, f) d\Xi. \tag{2.33}$$

This is called the compatibility condition [26, Chapter 2.3].

Despite the complexity of the full collision term, a simplification captures the overall behavior of the collision. The effect of many collisions is that the distribution will go towards the equilibrium, or in other words that the non-equilibrium part decays. This was modeled by Bhatnagar, Groos and Krook [32] as

$$\Omega(f, f) \approx -\frac{f^{neq}}{\tau} = -\frac{f - f^{eq}}{\tau}, \tag{2.34}$$

where τ is the mean time between collisions. This leads to the well known BGK-Boltzmann equation

$$\frac{\partial f}{\partial t} + \vec{u} \cdot \nabla f + \vec{g} \cdot \nabla_{\vec{u}} f = -\frac{f - f^{eq}}{\tau}. \tag{2.35}$$

The BGK-Boltzmann equation will be solved in this work to obtain solutions of the macroscopic flow equations, i.e. the Navier-Stokes equations.

From Eq. (2.25) follows that the conserved moments of the BGK collision operator vanish and, hence, that the BGK collision operator is conservative.

2.1.4 Formal solution of the BGK-Boltzmann equation

The simplification of the BGK collision operator allows for a formal solution of Eq. (2.35). This solution is usually credited to Kogan [33, Chapter 2.8.2], see e.g. [34, Chapter 4.2.2]. Other publications, such as e.g. [35, 7, 36] state this equation without reference. The derivation here is based on a lecture slide from a short course on *Gas-kinetic schemes for continuum and multiscale flows* given by Zhaoli Guo at the ICMMES 2018 conference.

First, the material derivative

$$\mathrm{D}_t f = \frac{\partial f}{\partial t} + \vec{u} \cdot \nabla f \tag{2.36}$$

is introduced. It accounts for the temporal change along characteristics. In terms of gas kinetics, it accounts for the temporal change of a single particle with constant velocity $\vec{u}$, rather than a fixed point in space. The particle path is called characteristics and is formulated by

$$\vec{x}' = \vec{x} - \vec{u}(t - t') \tag{2.37}$$

for a system in absence of gravitation forces. The location $\vec{x}'$ denotes the particle location at time t', which is assumed to be prior to t. The characteristics are the solution to the homogenous material derivative equation $\mathrm{D}_t f = 0$ with the solution $f(\vec{x}, \vec{u}, t) = f(\vec{x} - \vec{u}(t - t_0), \vec{u}, t_0)$.

Applying the material derivative to the BGK-Boltzmann equation yields the equation

$$\mathrm{D}_t f = -\frac{f - f^{eq}}{\tau}, \tag{2.38}$$

which should be solved for f. Hence, the equation is reordered, such that all occurrences of f appear on the left hand side:

$$\frac{1}{\tau} f + \mathrm{D}_t f = \frac{1}{\tau} f^{eq} \tag{2.39}$$

The next objective towards the solution is to unify the left hand side. It is observed that f appears once with a prefactor and once as a derivative. With the assumption of constant τ between t_0 and t, this is the result of a product rule of derivation with the second function missing. The second function must yield itself times $1/\tau$ after derivation with respect to t. This is satisfied by the exponential $\exp(t/\tau)$. Hence, the equation is multiplied with this function, such that

$$\frac{1}{\tau} e^{t/\tau} f + e^{t/\tau} \mathrm{D}_t f = \frac{1}{\tau} e^{t/\tau} f^{eq}. \tag{2.40}$$

Applying the inverse product rule of derivation,

$$\mathrm{D}_t \left(e^{t/\tau} f \right) = \frac{1}{\tau} e^{t/\tau} f^{eq} \tag{2.41}$$

brings the equation into a form that can be solved by simple integration along characteristics from an initial time t_0 to the current time t. On the left hand side the integration along the characteristics and the material derivate cancels out, such that the left hand side needs to be evaluated at t_0 and t. On the right hand side the integration remains:

$$\left(e^{t/\tau} f\right)_{t_0}^{t} = \frac{1}{\tau} \int_{t_0}^{t} e^{t'/\tau} f^{eq} dt'. \tag{2.42}$$

The integration along the characteristics leads to a transformation of the evaluation location, such that

$$e^{t/\tau} f(\vec{x}, \vec{u}, t) - e^{t_0/\tau} f(\vec{x} - \vec{u}(t - t_0), \vec{u}, t_0) = \frac{1}{\tau} \int_{t_0}^{t} e^{t'/\tau} f^{eq}(\vec{x} - \vec{u}(t - t'), \vec{u}, t') dt'. \tag{2.43}$$

Now the equation is solved for $f(\vec{x}, \vec{u}, t)$ by dividing by $e^{t/\tau}$. Note that $e^{t/\tau}$ is constant with respect to t', such that it can be pulled into the integral on the right hand side. The final formal solution of the BGK-Boltzmann equation reads

$$f(\vec{x}, \vec{u}, t) = e^{-(t-t_0)/\tau} f(\vec{x} - \vec{u}(t - t_0), \vec{u}, t_0) + \frac{1}{\tau} \int_{t_0}^{t} e^{-(t-t')/\tau} f^{eq}(\vec{x} - \vec{u}(t - t'), \vec{u}, t') dt'. \tag{2.44}$$

This formal solution is one key to the construction of the GKS, see [7]. It can be interpreted in the following way: The distribution at t depends on an initial state at t_0, which is traced back to the initial locations of the particles, and an equilibrium contribution. The initial state may contain a non-equilibrium part that decays along the characteristics. The attractor, i.e. the equilibrium, is not constant. Hence, it is also traced back for all times between t_0 and t and the attractor is the integral over all equilibrium states, weighting them by the exponential.

2.1.5 From the BGK-Boltzmann equation to Navier-Stokes

This work focuses on the flow evolution of continua on the macroscopic scale, while the prior sections dealt with the mesoscopic (and microscopic) scale. In this section it will be shown that the flow evolution equation of the mesoscopic scale, i.e. the BGK-Boltzmann equation (see Eq. (2.35)), yields the macroscopic equations of flow evolution, i.e. the Navier-Stokes equations, in a suitable limit. For this purpose the Navier-Stokes equations are derived from the BGK-Boltzmann equation. Note that in this work the term Navier-Stokes equations denotes the set of equations modeling mass, momentum and energy conservation. Other authors use the term Navier-Stokes exclusively for the momentum equation. Here, mass and energy equations are included to more convenient reference to the complete set of equations. In the process of the derivation the relation between viscosity and thermal diffusivity and the relaxation

time scale τ will be identified. The Chapman-Enskog expansion in this section is based on the GKS book by Xu [26, Appendix B].

In the remainder of this section a short hand notation is applied, such that $f_{,t} = \partial f/\partial t$ and similar for partial derivatives in space. Volume forces are ignored. The BGK-Boltzmann equation (in short hand notation) is

$$f_{,t} + uf_{,x} + vf_{,y} + wf_{,z} = -\frac{f - f^{eq}}{\tau}. \tag{2.45}$$

The relaxation time scale τ is split into a scale carrying quantity ϵ and a variation carrying quantity $\hat{\tau}$, such that $\tau = \epsilon\hat{\tau}$. The scale carrying quantity ϵ is constant with respect to the material derivative operator D_t. The variation carrying quantity is not constant.

Introducing the material derivative and the split of the relaxation time into the simplified BGK-Boltzmann equation and solving for f yields

$$f = f^{eq} - \epsilon\hat{\tau}\mathrm{D}_t f. \tag{2.46}$$

This differential equation can formally be solved by fixed point iteration, i.e. repetitive self insertion

$$f = f^{eq} - \epsilon\hat{\tau}\mathrm{D}_t\Big(f^{eq} - \epsilon\hat{\tau}\mathrm{D}_t\Big(f^{eq} - \epsilon\hat{\tau}\mathrm{D}_t\big(\dots\big)\Big)\Big). \tag{2.47}$$

Since ϵ is constant, it can be collected outside the material derivate. Expansion yields

$$f = f^{eq} - \epsilon\hat{\tau}\mathrm{D}_t f^{eq} + \epsilon^2\hat{\tau}\mathrm{D}_t\big(\hat{\tau}\mathrm{D}_t f^{eq}\big) - \epsilon^3\hat{\tau}\mathrm{D}_t\Big(\hat{\tau}\mathrm{D}_t\big(\hat{\tau}\mathrm{D}_t f^{eq}\big)\Big) + \dots. \tag{2.48}$$

Complementary to the above solution of the BGK-Boltzmann equation, it is possible to expand the distribution function in terms of the scale carrying quantity as

$$f = f_0 + \epsilon f_1 + \epsilon^2 f_2 + \epsilon^3 f_3 + \dots. \tag{2.49}$$

Equating coefficients of ϵ yields the sequence

$$\begin{aligned}
f_0 &= f^{eq} \\
f_1 &= -\hat{\tau}\left(f^{eq}_{,t} + uf^{eq}_{,x} + vf^{eq}_{,y} + wf^{eq}_{,z}\right) \\
f_2 &= \hat{\tau}^2 \quad \left(f^{eq}_{,tt} + 2uf^{eq}_{,tx} + 2vf^{eq}_{,ty} + 2wf^{eq}_{,tz}\right. \\
&\qquad\quad + u^2 f^{eq}_{,xx} + v^2 f^{eq}_{,yy} + w^2 f^{eq}_{,zz} \\
&\qquad\quad \left. + 2uvf^{eq}_{,xy} + 2uwf^{eq}_{,xz} + 2vwf^{eq}_{,yz}\right) \\
&\quad + \hat{\tau} \quad \left(\hat{\tau}_{,t} + u\hat{\tau}_{,x}\right)\left(f^{eq}_{,t} + uf^{eq}_{,x} + vf^{eq}_{,y} + wf^{eq}_{,z}\right) \\
&\quad \vdots
\end{aligned} \tag{2.50}$$

of non-equilibrium parts of f as functions of the equilibrium distribution f^{eq}.

The scale carrying quantity ϵ is proportional to the Knudsen number Kn for a given problem, where the Knudsen number is the ratio of particle mean free path and reference length. It is, hence, a measure for how good a system can be modeled with a macroscopic continuum's approach. Large Knudsen numbers mean that the gas is diluted. Inter particle collisions are rare and free advection is prevalent. Such a system can stay out of equilibrium for a long time. Small Knudsen numbers indicate that the system is much larger than the mean free path. Inter particle collisions are frequent, such that a system is driven towards equilibrium quickly. This is also observed in the above equation. For small ϵ, the distribution f is always close to equilibrium.

This work focuses on the continuum regime, i.e. low Knudsen numbers and, hence, small ϵ. The continuum regime comprises two sets of equations, namely the Euler and the Navier-Stokes equations. Both can be derived from the Chapman-Enskog expansion, as will be shown below.

First, the limit of zero Knudsen number, which implies that the mean free path is zero, is investigated. We consider shear flow. In general momentum is exchanged by particles penetrating other shear layers by free advection, before exchanging momentum by collision. Zero mean free path implies that no free advection takes place and, therefore, no momentum or energy is exchanged in a shear flow. This absence of diffusive transport is typical for the Euler equation. In fact, the Chapman-Enskog expansion to zeroth order (obtained for $\epsilon = 0$) yields the Euler equations.

Inserting the zeroth order Chapman-Enskog expansion $f = f^{eq}$ in Eq. (2.46) yields

$$f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} = 0, \tag{2.51}$$

where the right hand side cancels out. This equation means that the equilibrium state is transported with particle velocity. Taking the conserved moments of this equation and exchanging derivation and integration yields

$$\begin{aligned} &\int_\Xi \underline{\psi} \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} \right) d\Xi \\ &= \frac{\partial}{\partial t} \int_\Xi \underline{\psi} f^{eq} d\Xi + \frac{\partial}{\partial x} \int_\Xi \underline{\psi} u f^{eq} d\Xi + \frac{\partial}{\partial y} \int_\Xi \underline{\psi} v f^{eq} d\Xi + \frac{\partial}{\partial z} \int_\Xi \underline{\psi} w f^{eq} d\Xi = 0. \end{aligned} \tag{2.52}$$

Evaluating the moments according to Eq. (2.29) and Eq. (2.30) yields the Euler equations as

$$\frac{\partial}{\partial t} \begin{pmatrix} \rho \\ \rho U \\ \rho V \\ \rho W \\ \rho E \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} \rho U \\ \rho U^2 + p \\ \rho UV \\ \rho UW \\ U\left(\rho E + p\right) \end{pmatrix} + \frac{\partial}{\partial y} \begin{pmatrix} \rho V \\ \rho UV \\ \rho V^2 + p \\ \rho VW \\ V\left(\rho E + p\right) \end{pmatrix} + \frac{\partial}{\partial z} \begin{pmatrix} \rho W \\ \rho UW \\ \rho VW \\ \rho W^2 + p \\ W\left(\rho E + p\right) \end{pmatrix} = 0, \tag{2.53}$$

where the equation of state $p = \rho/(2\lambda)$ is used [7].

Next, we investigate the case of low but finite Knudsen numbers. In this case shear layer penetration takes place and momentum and energy are exchanged. For low Knudsen numbers terms with ϵ^2 can be neglected, such that the Chapman-Enskog expansion up to first order reads

$$f = f^{eq} - \tau \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} \right). \tag{2.54}$$

Inserting Eq. (2.54) into Eq. (2.46) and taking conserved moments yields

$$\int_{\Xi} \underline{\psi} \begin{pmatrix} (f^{eq} - \tau \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} \right))_{,t} \\ + u(f^{eq} - \tau \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} \right))_{,x} \\ + v(f^{eq} - \tau \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} \right))_{,y} \\ + w(f^{eq} - \tau \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,z} \right))_{,z} \end{pmatrix} d\Xi = 0, \tag{2.55}$$

where the right hand side cancels out due to the compatibility condition in Eq. (2.25). Collecting terms with τ on the right hand side yields the Euler equations on the left hand side and the diffusive terms on the right hand side, such that the Navier-Stokes equations are

$$\begin{aligned} &\int_{\Xi} \underline{\psi} \left(f^{eq}_{,t} + u f^{eq}_{,x} + v f^{eq}_{,y} + w f^{eq}_{,y} \right) d\Xi \\ &= \tau \int_{\Xi} \underline{\psi} \left(f^{eq}_{,tt} + 2u f^{eq}_{,tx} + 2v f^{eq}_{,ty} + 2w f^{eq}_{,tz} \right) d\Xi \\ &+ \tau \int_{\Xi} \underline{\psi} \left(u^2 f^{eq}_{,xx} + v^2 f^{eq}_{,yy} + w^2 f^{eq}_{,zz} + 2uv f^{eq}_{,xy} + 2uw f^{eq}_{,xz} + 2vw f^{eq}_{,yz} \right) d\Xi. \end{aligned} \tag{2.56}$$

Inserting the moments of the equilibrium distribution yields the Navier-Stokes equations in terms of conserved variables. The time derivatives on the right hand side must theoretically be replaced by self inserting the Navier-Stokes equations. Fortunately, the right hand side would than contain terms of the scale τ^2 that are neglected. Hence, the time derivatives can be replaced by the Euler Equations.

The detailed symbolic derivation is shown in Appendix E. Finally, the mass conservation equation is recovered as

$$\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x}(\rho U) + \frac{\partial}{\partial y}(\rho V) + \frac{\partial}{\partial z}(\rho W) = 0, \tag{2.57}$$

which is identical to the Euler mass equation. This is because in a single material system mass is not object to diffusive fluxes.

The momentum equations are

$$\begin{aligned}&\frac{\partial}{\partial t}(\rho U)+\frac{\partial}{\partial x}(\rho U^2+p)+\frac{\partial}{\partial y}(\rho UV)+\frac{\partial}{\partial z}(\rho UW)\\&=\frac{\partial}{\partial x}\left(2\mu\frac{\partial U}{\partial x}+\mu_2\nabla\cdot\vec{U}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial U}{\partial y}+\mu\frac{\partial V}{\partial x}\right)+\frac{\partial}{\partial z}\left(\mu\frac{\partial U}{\partial z}+\mu\frac{\partial W}{\partial x}\right),\end{aligned}\tag{2.58}$$

$$\begin{aligned}&\frac{\partial}{\partial t}(\rho V)+\frac{\partial}{\partial x}(\rho UV)+\frac{\partial}{\partial y}(\rho V^2+p)+\frac{\partial}{\partial z}(\rho VW)\\&=\frac{\partial}{\partial x}\left(\mu\frac{\partial U}{\partial y}+\mu\frac{\partial V}{\partial x}\right)+\frac{\partial}{\partial y}\left(2\mu\frac{\partial V}{\partial y}+\mu_2\nabla\cdot\vec{U}\right)+\frac{\partial}{\partial z}\left(\mu\frac{\partial V}{\partial z}+\mu\frac{\partial W}{\partial y}\right)\end{aligned}\tag{2.59}$$

and

$$\begin{aligned}&\frac{\partial}{\partial t}(\rho W)+\frac{\partial}{\partial x}(\rho UW)+\frac{\partial}{\partial y}(\rho VW)+\frac{\partial}{\partial z}(\rho W^2+p)\\&=\frac{\partial}{\partial x}\left(\mu\frac{\partial U}{\partial z}+\mu\frac{\partial W}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial V}{\partial z}+\mu\frac{\partial W}{\partial y}\right)+\frac{\partial}{\partial z}\left(2\mu\frac{\partial W}{\partial z}+\mu_2\nabla\cdot\vec{U}\right)\end{aligned}\tag{2.60}$$

for x, y and z directions, respectively. Finally, the energy equation is

$$\begin{aligned}&\frac{\partial}{\partial t}(\rho E)+\frac{\partial}{\partial x}(U(\rho E+p))+\frac{\partial}{\partial y}(V(\rho E+p))+\frac{\partial}{\partial z}(W(\rho E+p))\\&=\frac{\partial}{\partial x}\left(2\mu U\frac{\partial U}{\partial x}\ +\mu V\left(\frac{\partial V}{\partial x}+\frac{\partial U}{\partial y}\right)\ +\mu W\left(\frac{\partial W}{\partial x}+\frac{\partial U}{\partial z}\right)+\mu_2 U\nabla\cdot\vec{U}\ +\kappa\frac{\partial T}{\partial x}\right)\\&+\frac{\partial}{\partial y}\left(2\mu V\frac{\partial V}{\partial y}\ +\mu U\left(\frac{\partial U}{\partial y}+\frac{\partial V}{\partial x}\right)\ +\mu W\left(\frac{\partial W}{\partial y}+\frac{\partial V}{\partial z}\right)+\mu_2 V\nabla\cdot\vec{U}\ +\kappa\frac{\partial T}{\partial y}\right)\\&+\frac{\partial}{\partial z}\left(2\mu W\frac{\partial W}{\partial z}+\mu U\left(\frac{\partial U}{\partial z}+\frac{\partial W}{\partial x}\right)+\mu V\left(\frac{\partial V}{\partial z}+\frac{\partial W}{\partial y}\right)\ +\mu_2 W\nabla\cdot\vec{U}+\kappa\frac{\partial T}{\partial z}\right).\end{aligned}\tag{2.61}$$

These equations contain three material parameters that are related to diffusion. The viscosity is found as

$$\mu=\tau p,\qquad\text{and}\qquad\nu=\frac{\tau}{2\lambda}\tag{2.62}$$

for dynamic and kinematic viscosities, respectively. The usual viscosity is related to momentum transport by shear. Momentum can also be transported by compression and expansion, which is parametrized by the bulk viscosity

$$\zeta=\frac{2}{3}\frac{K}{K+3}\tau p.\tag{2.63}$$

In the Eqs. (2.58) to (2.61), the bulk viscosity is combined with the shear viscosity to form the second viscosity [37, Chapter 3]

$$\mu_2 = \zeta - \frac{2}{3}\mu, \tag{2.64}$$

which acts directly on the divergence $\nabla \cdot \vec{U} = U_{,x} + V_{,y} + W_{,z}$ of the velocity field. The third material parameter corresponds to heat diffusion. The thermal conductivity is found as

$$\kappa = \frac{K+5}{2} R\tau p. \tag{2.65}$$

Inserting the dynamic viscosity into the thermal conductivity shows that heat and momentum diffusion coefficients are coupled. The thermal diffusivity is $k = \kappa/(\rho c_p)$. Inserting the thermal conductivity κ from the above equation and the heat capacity c_p from equation Eq. (2.9) yields

$$k = \frac{K+5}{2} R\mu \frac{1}{\rho c_p} = \frac{K+5}{2} R\mu \frac{2}{(K+5)R} = \frac{\mu}{\rho} = \nu. \tag{2.66}$$

This is a well known defect of the BGK approximation. It implies that the Prandtl number $Pr = \nu/k = 1$ is fixed to one, which is not true for real gases.

2.2 On the history and variants of gas kinetic schemes

The term Gas Kinetic Scheme (GKS) describes numerical schemes for the simulation of fluid dynamics that are based on a finite volume discretization where fluxes are computed based on gas kinetic theory. In addition to that, other numerical methods that are based on gas kinetic theory, exist. In fluid dynamics, the Lattice Boltzmann Method (LBM) [9, 10, 11, 12] has seen a lot of attention and development in past three decades. It is mostly applied for fluid dynamics in the continuum's limit, i.e. at low Knudsen number, and for nearly incompressible flows. Recovering the energy equation in a monolithic LBM is memory extensive and computationally inefficient, as it requires large stencils [13, 38]. Alternatively, double distribution function approaches are possible [39]. For high Knudsen number flows the reference method is Direct Simulation Monte Carlo (DSMC) [40]. Therein, many particles are observed in the probabilistic setting of the Boltzmann equation. DSMC struggles in the low Knudsen number limit as shown in [41].

In contrast to LBM, GKS recovers the energy equation in a monolithic scheme with low memory consumption, enabling the usage of GKS for compressible flow simulations, such as super sonic flows or the simulation of fire induced flows. In comparison to DSMC, GKS can be formulated in a unified way that promises to solve all Knudsen number regimes with a single scheme.

The following sections deal with the origin and history of GKS and the many variants developed for different purposes.

2.2.1 Gas kinetic schemes for super sonic flows at low Knudsen numbers

One of the first ancestors of GKS is the beam scheme, published in 1974 by Sanders and Prendergast [42]. Therein, it is applied to astrophysical problems. It is assumed that every computational cell in an Eulerian framework contains gas in equilibrium. The Maxwellian equilibrium is then approximated by a linear ansatz, where particles move along a number of discrete beams, hence the name, with a single velocity. This approximation is constructed such that it conserves the macroscopic quantities. Depending on the velocity of the beam, certain amounts of mass, momentum and energy are transferred to adjacent cells. This method comprises the basic idea of GKS to construct fluxes from cell to cell based on a construction of a particle distribution function. In the beam scheme the distribution per cell was assumed constant, such that on a cell face between two cells, two fluxes are present. First, computed from the distribution in the left cell, a flux goes from left to right, and second, vise versa. By transporting conserved quantities, instantaneous relaxation is implied.

The central researcher of GKS and the author of its original publication is Kun Xu, a former Ph.D. student of the above mentioned Professor Kevin H. Prendergast. In his Ph.D. thesis from 1993 [43], Kun Xu developed a numerical scheme for the Navier-Stokes equation based on gas-kinetic theory that can be considered pre-GKS, as this term refers to an extended method of the one developed during his Ph.D. work. This 1993 scheme is published as one dimensional scheme by Prendergast and Xu [44] and as multidimensional scheme by Xu and Prendergast [45] for two-dimensional problems. The assumption of constant equilibrium distribution over a cell of the beam scheme is lifted. The fluxes $\underline{\mathcal{F}}$ in this scheme are computed as moments of a time dependent distribution function $f(t)$ on the cell face based on the formal BGK solution Eq. (2.44), i.e.

$$\underline{\mathcal{F}} = \int_{\Xi} \int u \underline{\psi} f(t) d\Xi dt. \tag{2.67}$$

The time dependent distribution function $f(t)$ depends on an initial distribution and a time dependent equilibrium, which both are functions of space. Both are modeled as Taylor expansion of the equilibrium, while the initial distribution is defined in parts, based on the velocity, with the right going particles based on the left equilibrium and the left going particles based on the right equilibrium on the cell face. The Taylor expansion parameters are computed from the gradients of the conserved variables. The expansions are

$$\begin{aligned} f^{eq}(x,t) &= f_0^{eq}(1 + \overline{a}x + \overline{A}t) \qquad \text{and} \\ f(x,0) \quad &= \begin{cases} f_l^{eq}(1 + a_l x) & \text{for} \quad x < 0 \\ f_r^{eq}(1 + a_r x) & \text{for} \quad x > 0, \end{cases} \end{aligned} \tag{2.68}$$

where $x = 0$ is the location of the cell face and $\overline{a}$, $\overline{A}$, a_l and a_r are the Taylor expansion coefficients. Left and right equilibria f_l^{eq} and f_r^{eq} are reconstructed from the flow field.

This 1993 scheme is tested for several super sonic flow cases in one and two dimensions and shows good shock capturing behavior.

Further incremental development can be found in the script of the lecture that Kun Xu held 1998 at the von Karman Institute for Fluid Dynamics [34]. The original GKS was published by Kun Xu in 2001 [7]. It introduces the name Gas-Kinetic Scheme (or rather Gas-Kinetic BGK Scheme, where the BGK is omitted in later publications forming the now common abbreviation GKS). In terms of the numerical scheme three advances are made with regard to the 1993 scheme. First, a rigorous reconstruction of the initial flow data around the interface based on the van Leer limiter [46] that is also used with classical shock capturing schemes, is applied. Second, a discontinuity is added to the expansion of the equilibrium distribution, such that

$$f^{eq}(x,t) = f_0^{eq}(1 + (1 - H(x))\overline{a}_l x + H(x)\overline{a}_r x + \overline{A}t), \tag{2.69}$$

where $H(x)$ denotes the Heaviside function. Finally, the initial distribution is first Taylor expanded before the initial equilibrium distribution is expanded to first order by Chapman-Enskog expansion (see Eq. (2.54)). By this the initial distribution also incorporates the (modeled) non-equilibrium part that is based on gradients of the macroscopic flow field. The resulting expansion is

$$f(x,0) = \begin{cases} f_l^{eq}(1 + a_l x - \tau(A_l + a_l u)) & \text{for} \quad x < 0, u < 0 \\ f_r^{eq}(1 + a_r x - \tau(A_r + a_l u)) & \text{for} \quad x > 0, u > 0. \end{cases} \tag{2.70}$$

In the evaluation of the scheme first the expansion coefficients of the initial distribution function a_l and a_r are computed from the respective limited gradients. Then the conserved moments of $f(x,0)$ are computed to obtain the flow state on the cell face. The expansion coefficients of the equilibrium distribution are then computed based on the gradients between the cell states and the computed cell face state. The temporal expansion coefficients A_l, A_r and $\overline{A}$ are computed based on the compatibility condition Eq. (2.25). This publication also reports a Prandtl number fix to set arbitrary Prandtl numbers by a posteriori modification of the energy flux. The scheme is tested for shock capturing tests in one and two dimensions, as well as viscous test, such as the laminar boundary layer.

This original GKS only takes normal derivatives at the cell faces into account. A multidimensional extension that takes normal gradients on the cell face into account was published subsequently by Xu et al. [47] in 2005. This publication also introduces an implicit time marching towards steady state solutions based on the LU-SGS methodology [48] for GKS. The multidimensional extension was further investigated by Li and Fu [49].

A theoretical analysis of the GKS was done by Ohwada [50]. The author found that the GKS for smooth flows is of Lax-Wendroff type and, hence, second-order accurate in space and time.

A drawback of the original GKS is the large computational effort in assembling the temporal expansion coefficients, the fluxes and the Prandtl number fix. Notable improvements were introduced by May et al. [51] in 2007. The computational cost

is reduced drastically by identifying similar terms in $f(t)$ that either cancel out, or vanish during moment integration due to the compatibility condition. Further, it is argued that the kinetic treatment of the tangential gradients is not necessary, and the corresponding terms in the stress tensor can be easily approximated from the gradients of the macroscopic flow field directly. Also, it is demonstrated how the Prandtl number can be incorporated into the spatial expansion coefficients. This reduces the computational cost drastically, since the original Prandtl number fix required kinetic computation of the heat flux, which comprises many computations. In May's approach the heat flux is corrected in the spatial gradient of the Maxwellian. By comparing Xu's computation of the expansion coefficient with the modified gradient of the Maxwellian, the authors find that only one expansion coefficient has to be divided by Pr to set the correct Prandtl number. Finally, the authors contribute to unstructured grid implementations of GKS by extrapolation the cell center flow states to obtain normal gradients for the computation of $\overline{a}_l$ and $\overline{a}_r$. Due to the vast simplifications in their improved scheme, they are able to obtain the first three-dimensional GKS results on a complex flow domain, i.e. transsonic flow around an complete aircraft.

Other authors have also implemented GKS on unstructured grids, see [16, 17, 18, 19]. Notable among these is a work of Ni et al. [16]. The construction of a time dependent distribution function allows things not possible in classical fluid dynamics. The authors compute the flow state as conserved moments of the time dependent distribution function at the end of a time step on the cell face and store it for the next time step. Hence, these face states can be used to compute the cell gradients and the face state is known such that computation from the initial distribution is not required [52]. In the work of Li et al. [18] hyper sonic flows at high temperatures with variable heat capacity are solved on unstructured grids for steady states utilizing the LU-SGS method.

A prevalent topic in computational fluid dynamics is the simulation of turbulent flows and turbulence modeling. Liao et al. [53] simulated nearly incompressible and fully compressible isotropic decaying turbulence with GKS in the framework of Direct Numerical Simulation (DNS) and compared directional splitting and multidimensional schemes. Kumar et al. [54] also simulated compressible turbulence and compare different reconstruction schemes. They find that the accuracy gain of Weighted Essentially Non-Oscillatory (WENO) reconstructions outweighs the increase in computational time compared to the van Leer limiter. The authors also find that at higher Mach numbers direct interpolation of temperature is better than interpolation of energy.

Many authors have added various Reynolds Averaged Naiver-Stokes (RANS) models to GKS [55, 17, 56, 19, 57]. Among these Li et al. [55] published both a direct simulation of a compressible mixing layer and a comparison of several RANS models. The scalar transport of turbulent properties is implemented directly into the GKS. A notable work is that of Righi [56, 58]. The author states that the incorporation of the turbulent viscosity in GKS can produce non-linear stress terms, while classical Navier-Stokes solvers can only reproduce linear terms. This results in much better capturing of the interaction between turbulent boundary layers and shocks. The work

of Li et al. [57] is notable because it implements GKS with a RANS model on a Cell-Vertex, rather than on a Cell-Centered grid.

Another trending topic in GKS development is high-order schemes. Recent years have seen several publications on high-order GKS. A first approach was published in 2008 by Li and Fu [59] for a GKS for smooth flows. Therein, a third-order flux function with pice wise parabolic reconstruction is introduced. The extension to flows with discontinuities followed in 2010 by Li et al. [60]. The initial and equilibrium distributions are Taylor expanded to second-order as compared to the first order in the original scheme. The order of accuracy is measured as four for advection of a pressure perturbation and three for a shock structure test. Further extensions of this scheme were published by Luo and Xu [61] and Liu and Tang [62]. Subsequently, Pan and Xu [63, 64] developed compact third-order schemes based on the slope update in [52]. Simultaneously, a two-stage fourth-order method, was published in [65, 66]. Further recent high-order schemes may be found in [67, 68, 69].

Volume forces can be included in GKS in the finite volume method by operator splitting. More correctly the volume force must also be included in the flux computation by considering them in the BGK equation. This procedure was published by Tian et al. [36]. Following this path, Luo et al. [70] developed a so-called well-balanced GKS, where momentum fluxes and source terms cancel out in steady state. Further development can be found in a preprint of Chen et al. [71].

This section shall be concluded with a GKS from 2013, published by Xuan and Xu [72] It features a continuous initial reconstruction that is based on the exact solution of the truncated Chapman–Enskog expansion of the kinetic BGK equation. This method is reported to have a much better computational efficiency, while keeping good shock capturing. It is the only variant of GKS, where a GPU implementation is reported [73].

2.2.2 Gas kinetic schemes for rarefied flows

The Navier-Stokes equations are the limit of the Boltzmann equation for vanishing Knudsen numbers. High Knudsen numbers describe rarefied flows, where the mean free path is on the order or larger than the characteristic length scale of the problem. In this case collisions are rare, but not necessarily negligible and the gas might be far from equilibrium. Such flows cannot be solved with classical CFD, as the Navier-Stokes equations do not hold for such cases. The mesoscopic scale and, hence, gas kinetics and the Boltzmann equation have to be used. Albeit being based on the Boltzmann equation, the methods mentioned in the prior section also do not qualify for rarefied flows. The reason is that they are based on truncated Chapman-Enskog expansions, which assume low Knudsen numbers. Modeling the initial non-equilibrium by the Chapman-Enskog expansion from gradients of the macroscopic flow field prevents large deviations from equilibrium. That is because an infinite number of non-equilibrium states yield the same macroscopic flow state. Hence, for simulation of rarefied flows explicit representation of the non-equilibrium particle distribution is required. For

this purpose the particle distribution must be discretized not only in space (as the macroscopic flow state) but also in velocity space. Disregarding these arguments, a continuums GKS for rarefied flows based on modifications of transport coefficients was published by Xu and Josyula [74] in 2006.

The major development of GKS for rarefied flows begins in 2010 with the publication of the first Unified Gas Kinetic Scheme (UGKS) by Xu and Huang [75]. The term unified means that the scheme targets the whole rage of Knudsen numbers from free molecular flow to the Navier-Stokes equations in an efficient manner in a single simulation. Such flows are relevant for instance in aero-space applications. The first step is introducing a discrete particle distribution function that is updated by a finite volume scheme. The time dependent distribution function on the cell faces is modeled following the same procedure as the original GKS [7] but for discrete velocities. The Chapman–Enskog model of the non-equilibrium is not required because the non-equilibrium is already encoded in the discrete particle distribution function. In addition to the discrete particle distribution function the macroscopic flow state is also tracked and used in defining the equilibrium distribution. Finally, the method comprises a flux based update scheme for the macroscopic variables, where an equilibrium part of $f(t)$ is integrated directly and the remainder (as it exists in a discrete manner) is integrated by quadrature, i.e. summation of the discrete terms over all discrete velocities. The update scheme for the discrete particle distribution function comprises a flux part that models the particle transport and a local (cell wise) BGK collision part. The scheme is tested for several one-dimensional test cases ranging from free molecular transport to classical shock capturing tests.

Following the initial publication many improvements have been proposed. Xu and Huang [76] replaced the BGK collision model by the Sharkov collision model that incorporates the correct Prandtl number. Multidimensional extension followed in 2012 by Huang et al. [41] and the same authors extended the scheme 2013 to micro flow simulations [77]. Liu and Xu [78] extended UGKS to multi-component plasma transport. A multigrid algorithm was applied to UGKS by Zhu et al. [79].

A reported deficiency of UGKS is that the local collision part in the update is not perfectly conservative. In a series of publications [80, 81, 82, 83, 84, 85] from the research group of Zhaoli Guo (in cooperation with Kun Xu) this deficiency is corrected. To this end advantages of LBM are incorporated into UGKS. The initial DUGKS was published in 2013 by Guo et al. [80] for low speed isothermal flows. In difference to UGKS, no formal BGK solution is applied in the update. The BGK equation is directly discretized, where the midpoint rule is applied for the convective term and trapezoidal rule for the collision. The trapezoidal rule makes the scheme implicit. The implicity can be solved symbolically by splitting the distribution function yielding an explicit scheme. Further, the equilibrium is Taylor expanded as usually done in LBM. The second paper of this publication series followed in 2015 by Guo et al. [81]. An unstructured grid implementation was published by Zhu et al. [82]. DUGKS was also applied to direct simulation of turbulent for both decaying three-dimensional Taylor-Green vortex and turbulence channel flow [83], particle-fluid flow based on

the Immersed Boundary Method (IBM) [84] and two phase flow based on the Chan-Hilliard equation [85].

2.2.3 Gas kinetic schemes for low Mach number flows at low Knudsen numbers

Even though GKS was developed with high Mach number and high Knudsen number flows in mind, low Mach number continuums flows were at all times subject of GKS publications. Even before the publication of the original GKS [7], it was used for low speed flows in two dimensions [14] and natural convection based on a double distribution function approach [86]. Both these simulations apply a simplified version of the GKS for smooth flow, i.e. flows without discontinuities/shocks. Without discontinuities a discontinuous reconstruction is not necessary and the initial and equilibriums distributions can be constructed as

$$\begin{aligned} f^{eq}(x,t) &= f_0^{eq}(1+ax+At) \qquad \text{and} \\ f(x,0) &= f_0^{eq}(1+ax-\tau(A+au)). \end{aligned} \tag{2.71}$$

It is obvious that only one set of expansion coefficients is present. This is because only a single gradient is present on the cell face. Due to the assumption of smooth flows the difference of the gradients in left and right hand side is negligible. This reduces the computational effort substantially. Liao et al. [53] applied this smooth GKS to the decay of isotropic turbulence and found that it works well for low Mach numbers. Jin et al. [87] implemented this scheme on a moving grid.

The simplification towards smooth reconstruction is not necessary. Chen et al. [88] investigated different reconstructions for smooth flows and found that a weak discontinuity in the reconstruction improves the robustness on under-resolved grids.

Wang and Guo [89] introduced an alternative GKS, which differs from the original GKS in the way that it does not employ the formal BGK solution. Instead of integrating a time dependent distribution function over one time step, this integration is approximated by one point Gauss integration, i.e. evaluation of the integrant in the middle of the interval. The distribution function at the middle of the time step is obtained by integration of the Boltzmann equation (without assumption of BGK at this point) along characteristics (i.e. particle paths) and the collision operator is approximated by a trapezoidal rule. This procedure is similar to the DUGKS procedure. Taylor expansion introduces the same expansion coefficients as in the original GKS. The trapezoidal rule makes this procedure formally implicit by requiring the equilibrium at the middle of the time step which can be solved by computing the state at this moment. This is possible due to the compatibility condition, which cancels out the unknown equilibrium. This procedure is reported to have superior stability compared to the original GKS.

Interesting smooth flows where compressibility plays a role occur in natural convection, where density differences drive the convection. Zhou et al. [90] used a GKS to simulate natural convection in a concentric annulus and Li and Wang [91] coupled IBM to an DUGKS to simulate natural convection around complex geometries. Further natural convection applications of DUGKS can be found in Wang et al. [92] and Wang et al. [93].

Finally, several comparisons between GKS variants and LBM were published. In 2003, Xu and He [94] compared a smooth isothermal GKS (no energy equation, constant temperature) to a simple single relaxation time LBM for very limited test cases. Guo et al. [95] compared a smooth GKS with energy equation to a multiple relaxation time LBM and found that, while LBM is about three times faster computationally, GKS has a larger modelling scope including high Mach number flows and thermal compressibility. The test cases in this publication are two-dimensional, but more complex as in the prior publication. The most recent comparison was published by Wang et al. [96] in 2016. In this study, a DUGKS is compared against a multiple relaxation time LBM for the simulation of decaying turbulence.

In the process of the work culminating in the present dissertation two intermediate results on GKS for natural convection were published. The first publication presents an unstructured grid implementation and performs several two-dimensional test cases, cf. Lenz et al. [15]. The second publication concerns an efficient implementation of GKS on massively parallel hardware (i.e. GPUs) and presents a Finite-Volume scheme on quad-tree like Cartesian grids with second-order conservative refinement, cf. Lenz et al. [97].

2.3 Derivation of the gas kinetic flux scheme at low Mach number under gravitational fields

In this section the multidimensional gas kinetic scheme for smooth flows is derived. This derivation is mainly based on the original GKS publication [7], its multidimensional extension [47] and the consideration of gravitational fields [36]. Starting from the formal BGK solution Eq. (2.44), a time dependent distribution function is constructed and flux densities are derived.

2.3.1 Integral conservation equation

Let

$$\underline{W} = (\rho, \rho\vec{U}, \rho E)^T \tag{2.72}$$

be a known flow state at $\vec{x} = \vec{0}$ and $t = 0$ in terms of conserved variables and let

$$\nabla\underline{W} \tag{2.73}$$

be the known spatial gradients of that flow state. The computation of these quantities depends on the finite volume implementation and is introduced in Section 2.4. The derivation of the fluxes in this section is independent of the finite volume details.

First, we consider the BGK-Boltzmann equation with gravitational field (see Eq. (2.35))

$$\frac{\partial f}{\partial t} + \vec{u} \cdot \nabla f + \vec{g} \cdot \nabla_{\vec{u}} f = -\frac{f - f^{eq}}{\tau}. \tag{2.74}$$

Taking conserved moments of this equation and integrating it over an arbitrary volume V yields

$$\int_V \frac{\partial \underline{W}}{\partial t} dV + \int_\Xi \underline{\psi} \vec{u} \cdot \left(\int_V \nabla f dV \right) d\Xi + \int_\Xi \int_V \underline{\psi} \left(\vec{g} \cdot \nabla_{\vec{u}} f \right) dV d\Xi = 0. \tag{2.75}$$

Using a corollary of the divergence theorem for the advective part transforms the volume integral to a surface integral, i.e.

$$\int_V \frac{\partial \underline{W}}{\partial t} dV + \int_\Xi \underline{\psi} \vec{u} \cdot \left(\int_{\partial V} \vec{n} f d\partial V \right) d\Xi + \int_\Xi \int_V \underline{\psi} \left(\vec{g} \cdot \nabla_{\vec{u}} f \right) dV d\Xi = 0, \tag{2.76}$$

where ∂V is the surface of V. Reordering the advective terms yields

$$\int_V \frac{\partial \underline{W}}{\partial t} dV + \underbrace{\int_{\partial V} \int_\Xi \underline{\psi} \left(\vec{u} \cdot \vec{n} \right) f d\Xi d\partial V}_{\text{Fluxes}} + \underbrace{\int_V \int_\Xi \underline{\psi} \left(\vec{g} \cdot \nabla_{\vec{u}} f \right) d\Xi dV}_{\text{Sources}} = 0. \tag{2.77}$$

The second and third term denote fluxes and sources, respectively. This leads to the integral conservation equation

$$\int_V \frac{\partial \underline{W}}{\partial t} dV + \int_{\partial V} \underline{\mathcal{F}} d\partial V = \int_V \underline{\mathcal{Q}} dV, \tag{2.78}$$

where

$$\underline{\mathcal{F}} = \int_\Xi \underline{\psi} \left(\vec{u} \cdot \vec{n} \right) f d\Xi \tag{2.79}$$

and

$$\underline{\mathcal{Q}} = -\int_\Xi \underline{\psi} \left(\vec{g} \cdot \nabla_{\vec{u}} f \right) d\Xi \tag{2.80}$$

are the flux densities projected on an arbitrary vector $\vec{n}$ with unit length (from here on called directional flux density) and volume sources, respectively [7, 36]. The source term $\underline{\mathcal{Q}}$ is investigated in Section 2.3.8.

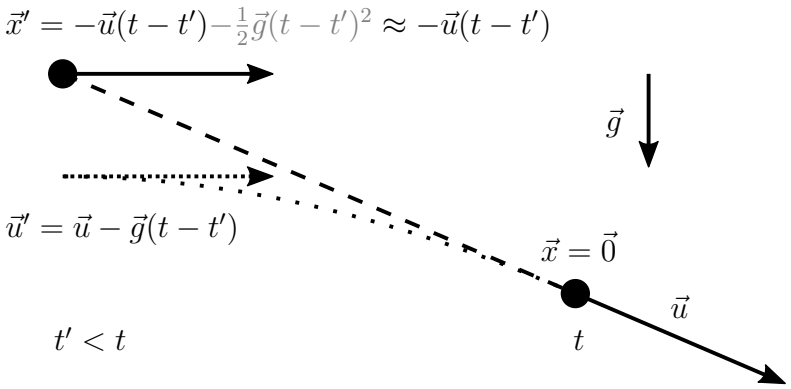


Figure 2.1: The characteristics follow the particles back in time. The gravitational field changes the particles velocity. The influence of the gravitational field on the particle trajectory is neglected [36]. The dotted line shows the particle trajectory with gravitational field and the dashed line the trajectory without gravitational field. The arrows are the velocity vectors.

2.3.2 Formal solution of the BGK-Boltzmann equation under gravitational fields

Now consider the formal BGK solution Eq. (2.44). The characteristics therein were constructed without gravitational fields, such that particles move along straight lines. We observe a particle motion under gravitational field at time t', location $\vec{x}'$ and with velocity $\vec{u}'$, where it is assumed that t' is prior to t. Applying constant acceleration to the particles leads to quadratic characteristics of the form [36]

$$\vec{x}' = \vec{x} - \vec{u}(t-t') - \frac{1}{2}\vec{g}(t-t')^2 \approx \vec{x} - \vec{u}(t-t'), \tag{2.81}$$

where the quadratic term in time can be neglected [36] and the time dependent velocity is

$$\vec{u}' = \vec{u} - \vec{g}(t-t'). \tag{2.82}$$

This procedure is also shown in Fig. 2.1. Evaluating the formal BGK solution Eq. (2.44) for these characteristics and assuming $\vec{x} = \vec{0}$ and $t_0 = 0$, yields [36]

$$f(\vec{0},\vec{u},t) = e^{-t/\tau} f(-\vec{u}t, \vec{u} - \vec{g}t, 0) + \frac{1}{\tau}\int_0^t e^{-(t-t')/\tau} f^{eq}(-\vec{u}(t-t'), \vec{u} - \vec{g}(t-t'), t')dt'. \tag{2.83}$$

2.3.3 Taylor expansion of the equilibrium

In order to capture the variation of the equilibrium distribution, f^{eq} is Taylor expanded in time, space and velocity space to first order, i.e.

$$\begin{aligned} f^{eq}(\delta\vec{x},\vec{u}+\delta u,\delta t) = f^{eq}(\vec{0},\vec{u},0) + \delta t\frac{\partial}{\partial t}f^{eq}(\vec{0},\vec{u},0) \\ + \delta\vec{x}\cdot\nabla f^{eq}(\vec{0},\vec{u},0) + \delta\vec{u}\cdot\nabla_{\vec{u}}f^{eq}(\vec{0},\vec{u},0). \end{aligned} \tag{2.84}$$

Factoring out the equilibrium distribution introduces the expansion coefficients $\vec{a}$, $\vec{b}$ and A, such that

$$f^{eq}(\delta\vec{x},\vec{u}+\delta u,\delta t) = f^{eq}(\vec{0},\vec{u},0)\left(1+\delta\vec{x}\cdot\vec{a}+\delta\vec{u}\cdot\vec{b}+\delta tA\right) \tag{2.85}$$

with

$$\vec{a} = \frac{\nabla f_0^{eq}}{f_0^{eq}} = \nabla \ln f_0^{eq}, \quad \vec{b} = \frac{\nabla_{\vec{u}} f_0^{eq}}{f_0^{eq}} = \nabla_{\vec{u}} \ln f_0^{eq} \quad \text{and} \quad A = \frac{1}{f_0^{eq}}\frac{\partial f_0^{eq}}{\partial t} = \frac{\partial \ln f_0^{eq}}{\partial t}, \tag{2.86}$$

where $f_0^{eq} = f^{eq}(\vec{0},\vec{u},0)$ is the equilibrium distribution corresponding to $\underline{W}$ [7].

Taking the logarithm of the equilibrium distribution Eq. (2.23) reveals the structure of the expansion coefficients as

$$\ln f_0^{eq} = \alpha_1 + u\alpha_2 + v\alpha_3 + w\alpha_4 + \frac{1}{2}\left(\vec{u}^2+\xi^2\right)\alpha_5 \tag{2.87}$$

with

$$\begin{aligned} \alpha_1 &= \ln\rho + \frac{K+3}{2}\ln\left(\frac{\lambda}{\pi}\right) - \lambda\vec{U}^2 \\ \alpha_2 &= 2\lambda U \\ \alpha_3 &= 2\lambda V \\ \alpha_4 &= 2\lambda W \\ \alpha_5 &= -2\lambda \end{aligned} \tag{2.88}$$

and $\underline{\alpha} = (\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5)^T$ [44]. It is evident that the logarithm of f_0^{eq} can be written as

$$\ln f_0^{eq} = \underline{\alpha}\cdot\underline{\psi} \tag{2.89}$$

and, hence, the expansion coefficients are [7]

$$\begin{aligned} \vec{a} &= (a_x,a_y,a_z)^T = \left(\underline{a}_x\cdot\underline{\psi},\underline{a}_y\cdot\underline{\psi},\underline{a}_z\cdot\underline{\psi}\right)^T \\ \vec{b} &= (b_x,b_y,b_z)^T = \left(\underline{b}_x\cdot\underline{\psi},\underline{b}_y\cdot\underline{\psi},\underline{b}_z\cdot\underline{\psi}\right)^T \\ \vec{A} &= \underline{A}\cdot\underline{\psi}. \end{aligned} \tag{2.90}$$

The expansion coefficients can now be obtained by directly taking gradients of $\ln f_0^{eq}$ and, hence, of the components of $\underline{\alpha}$ [95]. The gradients of the conserved variables are assumed to be known, e.g. for the time derivative

$$\frac{\partial \underline{W}}{\partial t} = \left(\frac{\partial \rho}{\partial t}, \frac{\partial \rho U}{\partial t}, \frac{\partial \rho V}{\partial t}, \frac{\partial \rho W}{\partial t}, \frac{\partial \rho E}{\partial t} \right)^T . \tag{2.91}$$

The computation of the time derivative will in fact be given in Section 2.3.5. The spatial gradients are computed from finite differences, see Section 2.4.3. From these gradients the gradients of the velocities and energy are obtained as

$$\begin{aligned} \frac{\partial \vec{U}}{\partial t} &= \left(\frac{1}{\rho} \frac{\partial \rho \vec{U}}{\partial t} \right) - \left(\frac{1}{\rho} \frac{\partial \rho}{\partial t} \right) \vec{U} \\ \frac{\partial E}{\partial t} &= \left(\frac{1}{\rho} \frac{\partial \rho E}{\partial t} \right) - \left(\frac{1}{\rho} \frac{\partial \rho}{\partial t} \right) \left(\frac{1}{2} \vec{U}^2 + \frac{K+3}{4\lambda} \right) . \end{aligned} \tag{2.92}$$

The expansion coefficients (spatial or temporal) are then obtained by taking gradients of $\underline{\alpha}$. They can efficiently be computed in backward order, here exemplary shown for the temporal expansion coefficient A:

$$\begin{aligned} A_5 &= 2 \frac{4\lambda^2}{K+3} \left(\frac{\partial E}{\partial t} - \vec{U} \cdot \frac{\partial \vec{U}}{\partial t} \right) \\ A_4 &= 2\lambda \frac{\partial W}{\partial t} - A_5 W \\ A_3 &= 2\lambda \frac{\partial V}{\partial t} - A_5 V \\ A_2 &= 2\lambda \frac{\partial U}{\partial t} - A_5 U \\ A_1 &= \left(\frac{1}{\rho} \frac{\partial \rho}{\partial t} \right) - A_2 U - A_3 V - A_4 W - \left(\frac{1}{2} \vec{U}^2 + \frac{K+3}{4\lambda} \right) A_5 \end{aligned} \tag{2.93}$$

Details on this derivation are given in Appendix A.1. The spatial expansion coefficients are obtained similarly.

The expansion coefficients in velocity space are also obtained by differentiation of $\ln f_0^{eq}$ [36]

$$\vec{b} = \nabla_{\vec{u}} \ln f_0^{eq} = 2\lambda \left(\vec{U} - \vec{u} \right) . \tag{2.94}$$

Alternatively, the expansion coefficients can be obtained by relating them to the macroscopic gradients, i.e.

$$\frac{\partial \underline{W}}{\partial t} = \int_\Xi \underline{\psi} A f_0^{eq} d\Xi = \int_\Xi \underline{\psi} (A \cdot \underline{\psi}) f_0^{eq} d\Xi = \int_\Xi \underline{\psi} \otimes \underline{\psi} f_0^{eq} d\Xi \; A. \tag{2.95}$$

The integral is a matrix, such that the expansion coefficients are solutions of a linear system of equations. The solution can be obtained symbolically and results in the same terms as Eq. (2.93) [45, 7, 26].

2.3.4 Time dependent distribution function

The initial distribution contains a non-equilibrium part and is hence first expanded to first order by the Chapman-Enskog expansion (see Eq. (2.54)) [7], including the volume force, such that

$$f(\delta\vec{x},\vec{u}+\delta\vec{u},0)=f^{eq}(\delta\vec{x},\vec{u}+\delta\vec{u},0)-\tau\left(\vec{u}\cdot\nabla f_0^{eq}+\vec{g}\cdot\nabla_{\vec{u}}f_0^{eq}+\frac{\partial f_0^{eq}}{\partial t}\right). \tag{2.96}$$

Second, the expanded equilibrium is inserted and the gradients of f_0^{eq} are evaluated [7] resulting in

$$f(\delta\vec{x},\vec{u}+\delta\vec{u},0)=f_0^{eq}\left(1+\delta\vec{x}\cdot\vec{a}+\delta\vec{u}\cdot\vec{b}-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right). \tag{2.97}$$

Finally, the expanded equilibrium Eq. (2.85) and expanded initial distribution Eq. (2.97) are inserted into the formal BGK solution Eq. (2.83). Solving the integral yields the time dependent distribution function [36]

$$f(\vec{0},\vec{u},t)=f_0^{eq}\left(1-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}\right)+(t-\tau)\,A\right), \tag{2.98}$$

also see detailed derivation in Appendix A.2.

2.3.5 Time derivative of conserved variables

Above, it is assumed that all derivatives of the conserved variables are known. The spatial derivatives can be obtained by e.g. finite differences on the grid (see Section 2.4). The time derivative can be obtained from the condition that the moments of the non-equilibrium part $f^{neq}=f-f^{eq}$ of a distribution must vanish [7]. Applying this to the first order Chapman-Enskog non-equilibrium

$$\int_{\Xi}\underline{\psi}\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)f_0^{eq}d\Xi=0 \tag{2.99}$$

and solving for the time derivative yields [7]

$$\frac{\partial\underline{W}}{\partial t}=\int_{\Xi}\underline{\psi}Af_0^{eq}d\Xi=-\int_{\Xi}\underline{\psi}\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}\right)f_0^{eq}d\Xi. \tag{2.100}$$

Inserting the collision invariants and expansion coefficients shows that the right hand side of this equation can be evaluated explicitly in the computation as large sum over moments of f_0^{eq} (cf. Section 2.1.2).

2.3.6 Directional flux densities

Finally, the directional flux densities are obtained by inserting Eq. (2.98) in Eq. (2.79), such that

$$\underline{\mathcal{F}} = \int_{\Xi} \underline{\psi} \left(\vec{u} \cdot \vec{n}\right) \left(1 - \tau \left(\vec{u} \cdot \vec{a} + \vec{g} \cdot \vec{b}\right) + \left(t - \tau\right) A\right) f_0^{eq} d\Xi. \tag{2.101}$$

Similar to the time derivative, the directional flux densities can be evaluated explicitly in the computation. For convenience the directional flux densities are split in three parts

$$\underline{\mathcal{F}} = \underline{\mathcal{F}}_1 - \tau \underline{\mathcal{F}}_2 + \left(t - \tau\right) \underline{\mathcal{F}}_3 \tag{2.102}$$

with

$$\begin{aligned}
\underline{\mathcal{F}}_1 &= \int_{\Xi} \underline{\psi} \left(\vec{u} \cdot \vec{n}\right) f_0^{eq} d\Xi, \\
\underline{\mathcal{F}}_2 &= \int_{\Xi} \underline{\psi} \left(\vec{u} \cdot \vec{n}\right) \left(\vec{u} \cdot \vec{a} + \vec{g} \cdot \vec{b}\right) f_0^{eq} d\Xi \quad \text{and} \\
\underline{\mathcal{F}}_3 &= \int_{\Xi} \underline{\psi} \left(\vec{u} \cdot \vec{n}\right) A f_0^{eq} d\Xi.
\end{aligned} \tag{2.103}$$

The first part $\underline{\mathcal{F}}_1$ contains advective fluxes, the second part $\underline{\mathcal{F}}_2$ contains diffusive fluxes and the third part $\underline{\mathcal{F}}_3$ is a correction for second order of convergence in time.

2.3.7 Prandtl number fix

One result of Section 2.1.5 is the unit Prandtl number $Pr = 1$ in the BGK approximation. Multiple solutions for this deficiency have been proposed [7, 51, 76]. In this work, the Prandtl number fix of the original GKS publication [7] is used. The idea is to subtract the diffusive heat flux density q from the energy flux density $\mathcal{F}^{\rho E}$ and add q/Pr, i.e.

$$\mathcal{F}^{\rho E}_{(Pr \text{ fixed})} = \mathcal{F}^{\rho E} + \left(\frac{1}{Pr} - 1\right) q. \tag{2.104}$$

The diffusive heat flux can also be obtained as moment of the time dependent distribution function f, such that

$$q = \frac{1}{2} \int_{\Xi} \left(\left(\vec{u} - \vec{U}\right) \cdot \vec{n}\right) \left(\left(\vec{u} - \vec{U}\right)^2 + \xi^2\right) f d\Xi. \tag{2.105}$$

After inserting Eq. (2.98) and simplification the heat flux is

$$q = -\tau \left(\mathcal{F}_2^{\rho E} + \mathcal{F}_3^{\rho E} - \vec{U} \cdot \left(\vec{\mathcal{F}}_2^{\rho \vec{U}} + \vec{\mathcal{F}}_3^{\rho \vec{U}}\right)\right). \tag{2.106}$$

2.3.8 Volume forces

In Eq. (2.80) the kinetic description of the volume source term is given in the form

$$\underline{\mathcal{Q}} = -\int_{\Xi} \underline{\psi}\,(\vec{g}\cdot\nabla_{\vec{u}} f)\,d\Xi. \tag{2.107}$$

This formula is difficult to interpret due to the gradient in velocity space $\nabla_{\vec{u}}$. Further, an unknown f complicates the interpretation. Fortunately, f is not arbitrary but a probability distribution density function in particle velocity space $\vec{u}$, such that $f \geq 0$. From the physical consideration of finite momentum and energy follows that particles have a finite velocity, i.e.

$$\lim_{u\to\pm\infty} f = 0. \tag{2.108}$$

This implies

$$\lim_{u\to\pm\infty} f_{,u} = \lim_{u\to\pm\infty} f_{,uu} = ... = 0 \tag{2.109}$$

and based on L'Hospital's rule

$$\lim_{u\to\pm\infty} uf = \lim_{u\to\pm\infty} u^2 f = ... = 0. \tag{2.110}$$

From Eq. (2.108) follows that

$$\int_{-\infty}^{\infty} f_{,u} du = 0. \tag{2.111}$$

Evaluating the the first component of $\underline{\mathcal{Q}}$ under the assumption of constant $\vec{g}$ over $\vec{u}$ yields

$$\mathcal{Q}^{\rho} = -\int_{\Xi} (g_x f_{,u} + g_y f_{,v} + g_w f_{,w})\,d\Xi = 0. \tag{2.112}$$

The first momentum component is

$$\mathcal{Q}^{\rho U} = -\int_{\Xi} (g_x u f_{,u} + g_y u f_{,v} + g_w u f_{,w})\,d\Xi = -\int_{\Xi} g_x u f_{,u} d\Xi, \tag{2.113}$$

where the mixed terms vanish due to Eq. (2.111). Partial integration gives

$$\mathcal{Q}^{\rho U} = g_x \int_{\Xi} f d\Xi - \int_{v\otimes w} \underbrace{(uf)_{u=-\infty}^{\infty}}_{\substack{=0 \\ \text{Eq. (2.110)}}} dvdw = g_x \int_{\Xi} f d\Xi \tag{2.114}$$

and, hence, the generalization is

$$\mathcal{Q}^{\rho\vec{U}} = \vec{g} \int_{\Xi} f d\Xi, \tag{2.115}$$

which can readily be interpreted as

$$\mathcal{Q}^{\rho\vec{U}} = \vec{g}\rho. \tag{2.116}$$

Similarly, the energy volume source is obtained after partial integration as

$$\mathcal{Q}^{\rho E} = \vec{g} \cdot \int_{\Xi} \vec{u} f d\Xi. \tag{2.117}$$

In the present work three schemes for volume forces are implemented. They are introduced below:

Consistent source term treatment (1): The first scheme is based on the work of Tian et al. [36]. The momentum source is used as given in Eq. (2.116), i.e.

$$\mathcal{Q}^{\rho\vec{U}} = \vec{g}\rho. \tag{2.118}$$

The integral for the energy source in Eq. (2.117) can be interpreted as momentum, as well as flux density of mass. For consistent source term treatment, the latter has to be implemented [36] as

$$\mathcal{Q}^{\rho E} = \vec{g} \cdot \vec{\mathcal{F}}_i^{\rho}, \tag{2.119}$$

where $\vec{\mathcal{F}}_i^{\rho}$ is the mass flux density of the i-th cell, which is computed as average of the adjacent cells, i.e.

$$\vec{\mathcal{F}}_i^{\rho} = \frac{1}{6} \sum_{j=1}^{6} \mathcal{F}_j^{\rho} \vec{n}_j. \tag{2.120}$$

Consistent source term treatment applied on density variation (2): The energy update of the second scheme is similar to the first one, but the momentum update is changed, such that the force is only applied to a density variation. A reference density ρ_0 is specified and the momentum source is

$$\mathcal{Q}^{\rho\vec{U}} = \vec{g}(\rho - \rho_0). \tag{2.121}$$

Constant temperature energy source applied on density variation (3): The third scheme is based on the observation that a volume force shall not alter the temperature. Hence, first the temperature is computed from the old momentum, i.e.

$$\lambda = \lambda(\rho\vec{U}) \tag{2.122}$$

Then the momentum is updated

$$\mathcal{Q}^{\rho\vec{U}} = \vec{g}(\rho - \rho_0). \tag{2.123}$$

Finally, the energy is computed based on the temperature and the updated momentum, i.e.

$$E = E\left(\lambda, \rho\vec{U} + \int \mathcal{Q}^{\rho\vec{U}} dt\right). \tag{2.124}$$

2.3.9 Component transport

For many applications fluid simulations are augmented by component transport equations of advection-diffusion type. For the simulation of fire, the fluid contains multiple species. This can be implemented into GKS by directly tracking multiple partial densities [98, 99]. Alternatively, the species can be determined by their mixture where the fraction of the species is tracked as passive scalar. A passive scalar transport extension of GKS was published in 2005 by Li et al. [100]. This approach comes out to be easily added to an existing GKS.

Let Θ be the mass specific passive scalar. Then $\rho\Theta$ denotes the volume specific amount. The tuple of conserved variables including the passive scalar is

$$\underline{W} = \left(\rho, \rho\vec{U}, \rho E, \rho\Theta\right)^T, \tag{2.125}$$

which is generated by the collision invariants

$$\underline{\psi} = \left(1, \vec{u}, \frac{1}{2}(\vec{u}^2 + \xi^2), \theta\right), \tag{2.126}$$

where θ is a fictional internal degree of freedom related to the passive scalar Θ. The introduction of θ implies that the equilibrium is to be modified as

$$f^{eq} = \rho\left(\frac{\lambda}{\pi}\right)^{\frac{K+4}{2}} \exp\left(-\lambda\left(\left(\vec{u} - \vec{U}\right)^2 + \xi^2 + (\theta - \Theta)^2\right)\right). \tag{2.127}$$

Based on this modified equilibrium a method including the passive scalar is derived in [100]. It is further shown that the addition of the passive scalar does not change the basic flow solver. The expansion coefficients

$$A_\theta = 2\lambda\frac{\partial\Theta}{\partial t} \qquad \text{with} \qquad \frac{\partial\Theta}{\partial t} = \left(\frac{1}{\rho}\frac{\partial\rho\Theta}{\partial t}\right) - \left(\frac{1}{\rho}\frac{\partial\rho}{\partial t}\right)\Theta \tag{2.128}$$

and similarly $\vec{a}_\theta$ are derived. The passive scalar flux is then found to be [100]

$$\mathcal{F}^{\rho\theta} = \Theta\mathcal{F}^\rho - \tau_D p\int_\Xi (\vec{u}\cdot\vec{n})(\vec{u}\cdot\vec{a}_\theta) f_0^{eq} d\Xi + (t - \tau_D)p\int_\Xi (\vec{u}\cdot\vec{n}) A_\theta f_0^{eq} d\Xi, \tag{2.129}$$

where $p = \rho/(2\lambda)$ is the pressure and $\tau_D = 2\lambda D$ with D being the diffusivity of the passive scalar. This procedure allows to use arbitrary Schmidt numbers $Sc = \nu/D$, as long as stability issues are not taken into account.

2.4 Finite volume implementation on uniform Cartesian grids

2.4.1 Finite volume update equation

The finite volume method is a discretization for partial differential equations. The essence of this method is to account for the solution $\underline{W}$ of a partial differential equation

in conservation formulation (with source terms), i.e.

$$\frac{\partial \underline{W}}{\partial t} + \nabla \cdot \underline{\vec{\mathcal{F}}} = \underline{\mathcal{Q}}, \tag{2.130}$$

where $\underline{\vec{\mathcal{F}}}$ denotes the flux density and $\underline{\mathcal{Q}}$ the volume forces. Integrating this equation over an arbitrary volume V and utilizing the divergence theorem yields the integral conservation equation [101, Chapter 2.1.1]

$$\int_V \frac{\partial \underline{W}}{\partial t} dV + \int_{\partial V} \vec{n} \cdot \underline{\vec{\mathcal{F}}} d\partial V = \int_V \underline{\mathcal{Q}} dV, \tag{2.131}$$

where ∂V is the boundary of V and $\vec{n}$ is the outward facing normal on every point on the volume boundary ∂V. In the gas kinetic context this form of the Boltzmann equation is derived in Section 2.3.1 with $\vec{n} \cdot \underline{\vec{\mathcal{F}}} = \underline{\mathcal{F}}$ and Sections 2.3.6 and 2.3.8 introduce the kinetic models for $\underline{\mathcal{F}}$ and $\underline{\mathcal{Q}}$, respectively.

The integral conservation equation holds for every volume. For practical computations, it is solved on a limited domain with artificial boundary conditions. The change of $\int_V \underline{W} dV$ over time, hence, depends only on the fluxes over the domain boundaries and sources in the domain. For practical purpose not only the global conservation is of interest but also the local flow states. For this reason, the domain is split into many finite volumes V_i [102, Chapter 2.6.2] (hence the name finite volume method), which are called cells, such that

$$V = \bigcup_i V_i. \tag{2.132}$$

The integral conservation equation must also hold for all cells, i.e.

$$\int_{V_i} \frac{\partial \underline{W}}{\partial t} dV_i + \int_{\partial V_i} \vec{n} \cdot \underline{\vec{\mathcal{F}}} d\partial V_i = \int_{V_i} \underline{\mathcal{Q}} dV_i. \tag{2.133}$$

For the discretization of this equation the solution $\underline{W}$ and the source term $\underline{\mathcal{Q}}$ are averaged over a cell, such that

$$\underline{W}_i = \frac{1}{V_i} \int_{V_i} \underline{W} dV_i \tag{2.134}$$

and

$$\underline{\mathcal{Q}}_i = \frac{1}{V_i} \int_{V_i} \underline{\mathcal{Q}} dV_i. \tag{2.135}$$

Further, the cells are constructed in a way that boundaries consist of regular plane faces A_j, such that the boundary integration of the fluxes can be split into integrals over multiple cell faces, which are approximated by one point Gauss integration, i.e. evaluation of the flux densities at the cell face center. Time integration over one time step, i.e. from 0 to Δt, then yields the finite volume update equation

$$\underline{W}_i^{n+1} = \underline{W}_i^{n+1} - \frac{1}{V_i} \sum_j A_j \int_0^{\Delta t} \underline{\mathcal{F}}_j dt + \Delta t \underline{\mathcal{Q}}_i, \tag{2.136}$$

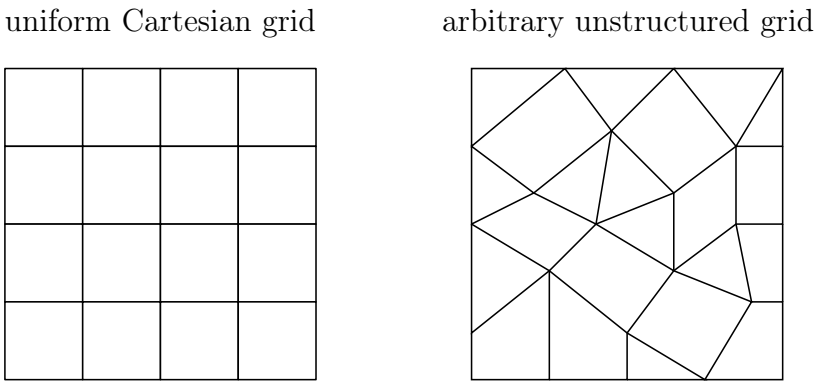


Figure 2.2: The left figure shows a uniform Cartesian grid (which is mainly considered in this work) and the right figure shows an arbitrary unstructured grid. (For visualization only 2D grids are shown.)

where the source term is integrated with the simple Euler forward method. The directional flux density $\underline{\mathcal{F}}$ can be integrated in time analytically, as it depends on the time dependent distribution function Eq. (2.98). The integral is

$$\int_0^{\Delta t} \underline{\mathcal{F}} dt = \Delta t \underline{\mathcal{F}}_1 - \tau \Delta t \underline{\mathcal{F}}_2 + \left(\frac{1}{2} \Delta t^2 - \tau \Delta t \right) \underline{\mathcal{F}}_3. \tag{2.137}$$

The conservation property of the finite volume method stems from the fact that fluxes are computed per cell face and a cell face is always shared between two cells. A flux that is subtracted from one cell in Eq. (2.136) will appear with opposite sign for the adjacent cell. By this, no amounts of the conserved variables can appear or disappear and the method is conservative by construction.

2.4.2 Grid layout

The finite volume update equation Eq. (2.136) holds for arbitrary cell shapes as long as the cell faces are planar. It is used for grid layouts that can be distinguished by structured/unstructured, uniform/non-uniform, Cartesian/Non-Cartesian, cell-centered/vertex-centered and more.

In this work two types of grids are considered, as shown in Fig. 2.2. The main focus lies on uniform Cartesian grids with unstructured connectivity. Further, some results of an early implementation in two dimensions on an arbitrary unstructured grid will be shown.

For the uniform Cartesian grid all cells are cubes with the same side length Δx and, hence, $V_i = \Delta x^3$ and $A_j = \Delta x^2$. Since all cells are equal, no individual cell geometry information is required. Each cell has 6 cell faces, such that $j = 1, \ldots, 6$ in Eq. (2.136). For regular domains structured grids are optimal, as they do not

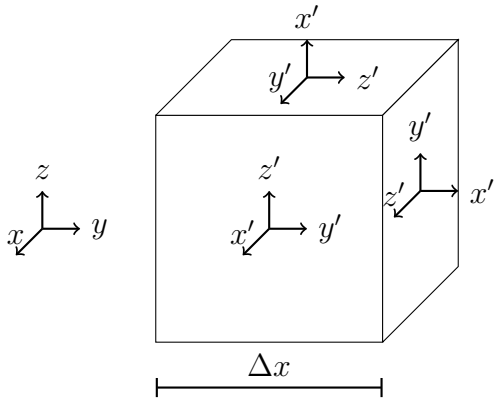


Figure 2.3: Rotated coordinate systems for the flux computation.

require any connectivity information. Since this work targets on irregular domains an unstructured connectivity is used. On structured grids, indices for all space directions exist and neighbor cells can be addressed by incrementing/decrementing these indices. This implies that the whole domain is covered by cells. On irregular domains this might involve large overheads in terms of memory consumption. Hence, only the cells that are inside the flow domain are stored. For connectivity, the neighbor cells have to be stored explicitly. In three dimensions this requires storing of indices to six neighbor cells. Edge and corner neighbors can be addressed by single and double pointer chasing, i.e. following first the connectivity to a face neighbor, then going in another direction to a face neighbor of this neighbor cell, which is an edge neighbor of the first cell. On arbitrary unstructured grids, both individual cell geometry (volume, cell center location, face normals) and connectivity to all surrounding cells have to be stored.

From a perspective of grid generation, Cartesian grids are more straight forward to generate automatically, while arbitrary unstructured grids usually fit better to the shape of the flow domain. On Cartesian grids, realizing inclined or curved walls with more then first order geometric accuracy is not trivial and requires special methods.

2.4.3 Flow field reconstruction around cell faces

The evaluation of the directional flux densities requires the flow state and its gradients on the cell faces. The directional flux densities are also formulated in terms of an arbitrary normal vector $\vec{n}$ and arbitrary particle velocity $\vec{u}$. On a Cartesian grid, only six options for normal vectors exist. This number is reduced to three, by considering that a cell face with negative normal vector is identical to a cell face with positive normal vector viewed from the adjacent cell. In order to prevent the necessity to implement the flux computation three times, the flow state and its gradients are rotated to a local coordinate system (x', y', z'), where the new axis x' is identical to the normal of the cell face $\vec{n}$. Transforming the flow state and its gradients reduces the term $(\vec{n} \cdot \vec{u})$ to just u, i.e. the particle velocity normal to the cell face. On Cartesian grids, the rotation can be computed by a simple swapping of coordinate axes. Here,

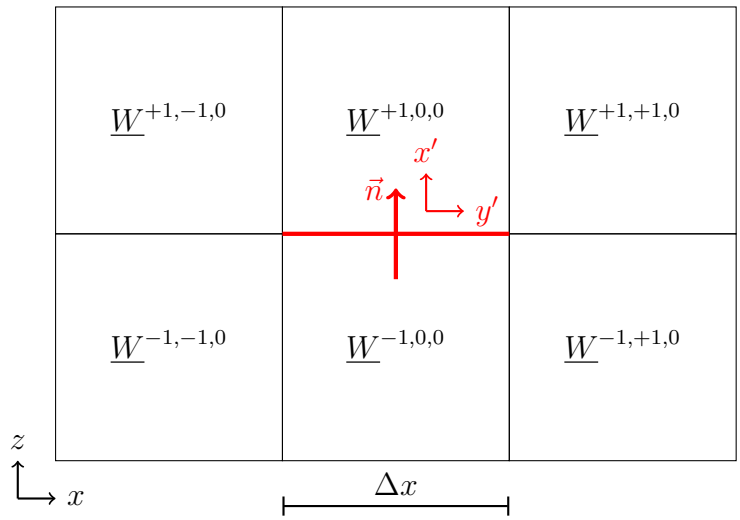


Figure 2.4: Stencil for computation of the hydrodynamic variables and their gradients on the cell face. The current cell face is marked in red and the cells are named as $\underline{W}^{x',y',z'}$ relative to the current cell face in the local coordinate system on the cell face.

the rotation is chosen in such a way that all cell face coordinate axes point in positive directions of the original coordinate system, see Fig. 2.3. Let $\underline{\underline{R}}$ be the transformation matrix to transform from the global coordinate system to a local coordinate system, then

$$(\rho\vec{U})' = \underline{\underline{R}}(\rho\vec{U}) \tag{2.138}$$

and, hence,

$$\left(\vec{\mathcal{F}}^{\rho\vec{U}}\right) = \underline{\underline{R}}^T \left(\vec{\mathcal{F}}^{\rho\vec{U}}\right)'. \tag{2.139}$$

The gradients are computed directly in the local coordinate system (denoted by the operator ∇'), such that the gradient operator requires no transformation.

On a uniform Cartesian grid, the flow state on the cell face is obtained as average of the two adjacent cells

$$\underline{W} = \frac{\underline{W}^{+1,0} + \underline{W}^{-1,0}}{2} \tag{2.140}$$

following the nomenclature in Fig. 2.4. This operation is second-order accurate in Δx. The gradients are obtained by central differences also with second order. The reason for this is that the cell face center point is directly in the center of the stencil, as seen in Fig. 2.4. The normal gradient is computed directly between the cell states on the positive and the negative side of the cell face. Tangential gradients are obtained by first averaging the two edge neighbors in positive and negative direction and then

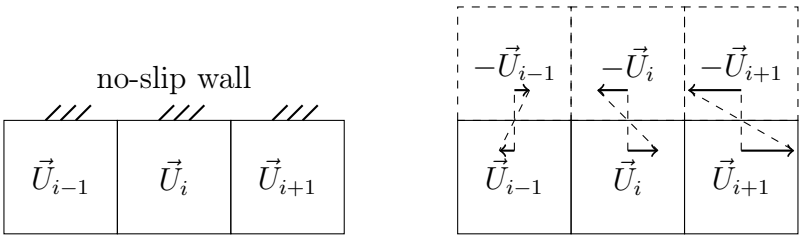


Figure 2.5: No-slip boundary condition: The artificial no-slip boundary condition at a solid wall is implemented via ghost cells (dashed). The velocities for the no-slip wall are inverted to achieve zero velocity directly on the wall. The no-slip wall is a special case of the moving walls with zero velocity.

taking the finite difference. Hence, the gradient is computed by

$$\nabla' \underline{W} = \frac{1}{\Delta x} \begin{pmatrix} \underline{W}^{+1,0,0} - \underline{W}^{-1,0,0} \\ \frac{1}{4}\left(\underline{W}^{+1,+1,0} + \underline{W}^{-1,+1,0} - \underline{W}^{+1,-1,0} - \underline{W}^{-1,-1,0}\right) \\ \frac{1}{4}\left(\underline{W}^{+1,0,+1} + \underline{W}^{-1,0,+1} - \underline{W}^{+1,0,-1} - \underline{W}^{-1,0,-1}\right) \end{pmatrix}. \tag{2.141}$$

It is worth noting that this assumes that the flow state in the cell is located in the cell center, which is different from the assumption of cell averaged flow states, see Eq. (2.134). The difference between the two assumptions is of second order in Δx because the integral in Eq. (2.134) can be approximated by one point Gauss integration, which is second-order accurate.

2.5 Boundary Conditions

The solution of a partial differential equation on finite domains is always an approximation due to their artificial boundary conditions that model the influence of the rest of the world. Here, the boundary conditions are implemented in the framework of the finite volume method. The idea is to use (nearly) the same flux computation on the artificial boundaries as in the fluid domain. For this purpose, ghost cells are introduced around the fluid domain [7]. The values of these ghost cells are set to enable the flux computation to compute the correct fluxes. An example of this procedure is shown in Fig. 2.5 for a no-slip wall. Several boundary conditions are introduced in the following sections.

Boundary fluxes are fixed in the flux computation to ensure conservation. If the boundary condition is labeled as a solid wall, the mass flux (and the passive scalar flux) is set to zero. If a boundary is labeled as insulated, the energy flux is set to zero.

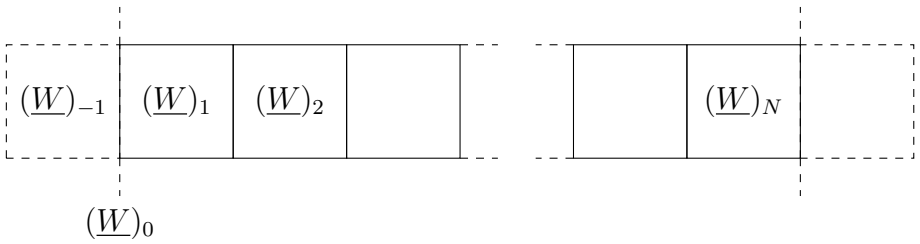


Figure 2.6: Naming conventions for the cells at the boundaries: The boundary ghost cell with index -1 is located outside the domain. The fluid cells with indices 1 and 2 are the first and second cells from the boundary, respectively, and the cell with index N is the last cell on the opposite side of the domain directly inside the domain. The index 0 denotes values directly on the boundary.

2.5.1 Walls

Common boundary conditions in fluid flow simulations are solid walls, which are not penetrated by the fluid. With regard to the velocity field, these walls are no-slip boundary conditions, meaning the relative velocity between the wall and the fluid on the wall vanishes. This boundary condition is of Dirichlet type. It is implemented by [7]

$$(\vec{U})_{-1} = 2(\vec{U})_0 - (\vec{U})_1, \tag{2.142}$$

where the indices -1, 0 and 1 denote ghost cell, wall and first fluid cell, see Fig. 2.6 for naming conventions. This equation assumes that the cell center of the fluid cell has the same distance from the wall as the center of the ghost cell, which is directly satisfied on a uniform Cartesian grid.

When modeling the pressure field in the vicinity of the wall we neglect the viscous terms in the Navier-Stokes equations and look at the one dimensional Euler equation

$$\frac{\partial(\rho U)}{\partial t} + \frac{\partial}{\partial x}\left(\rho U^2 + p\right) = 0 \qquad \text{with} \qquad U(x=0) = 0 \tag{2.143}$$

and x being the wall normal. The one dimensional Euler equation stems from neglecting the derivatives in the other space directions. This can be done for straight walls, but not for curved walls. Expanding the derivatives and inserting the vanishing velocity at the wall yields

$$\frac{\partial p}{\partial x} = 0. \tag{2.144}$$

This zero gradient in pressure is set in terms of density

$$(\rho)_{-1} = 2(p)_1(\lambda)_{-1} \qquad \text{with} \qquad (p)_1 = \frac{(\rho)_1}{2(\lambda)_1}. \tag{2.145}$$

The assumption of zero pressure gradient at the wall does not hold under gravitational fields. Hence, also a linear pressure extrapolation based on a second fluid cell with

index 2 is implemented where

$$(\rho)_{-1} = 2(p)_{-1}(\lambda)_{-1} \qquad \text{with} \qquad (p)_{-1} = 2(p)_1 - (p)_2. \tag{2.146}$$

The temperature (in terms of λ) at the wall can be modeled by both Dirichlet and Neumann type boundary conditions for isothermal and insulated walls, respectively. For insulated walls, the temperature has a zero gradient, such that

$$(\lambda)_{-1} = (\lambda)_1. \tag{2.147}$$

For the isothermal wall the temperature is treated similar to the velocity at the wall [7], such that

$$(\lambda)_{-1} = 2(\lambda)_0 - (\lambda)_1. \tag{2.148}$$

The passive scalars at walls are also modeled by zero gradient boundary conditions.

2.5.2 Inflow

At inflow boundary conditions, flow enters the domain with a prescribed velocity and temperature. Hence, velocity and temperature in the ghost cells are set by equations Eq. (2.142) and Eq. (2.148), respectively. The pressure, which is unknown at the inflow, is set based on the zero gradient in Eq. (2.145). The difference to wall boundary conditions is that mass flux is (obviously) non-zero at an inflow boundary.

2.5.3 Outflow

In comparison to the inflow, the outflow boundary is characterized by a prescribed pressure that is converted to a density by

$$(\rho)_0 = 2(p)_0(\lambda)_0 \qquad \text{with} \qquad (\lambda)_0 = \frac{(\lambda)_{-1} + (\lambda)_1}{2}, \tag{2.149}$$

where the temperature on the boundary is interpolated between the domain cell and the ghost cell. The density in the ghost cell is

$$(\rho)_{-1} = 2(\rho)_0 - (\rho)_1. \tag{2.150}$$

The velocities are linearly extrapolated by

$$(\vec{U})_{-1} = 2(\vec{U})_1 - (\vec{U})_2. \tag{2.151}$$

In the basic version of this boundary condition inflow is theoretically allowed but not necessarily stable. In this version, the temperature is extrapolated by

$$(\lambda)_{-1} = 2(\lambda)_1 - (\lambda)_2. \tag{2.152}$$

In the advanced version, the temperature is modeled by zero gradient, see Eq. (2.147). Further, inflow is prohibited by setting the velocity to zero if the extrapolated velocity points inwards, i.e.

$$(\vec{U})_{-1} = \begin{cases} 0 & \text{for} \quad (2(\vec{U})_1 - (\vec{U})_2) \cdot \vec{n} < 0 \\ 2(\vec{U})_1 - (\vec{U})_2 & \text{else} \end{cases} \tag{2.153}$$

with $\vec{n}$ being the outward pointing normal on the boundary. This version is used when outflow cannot be assured.

2.5.4 Open boundary

Open boundaries in this work are boundaries that allow unprescribed inflow but no outflow. For the open boundary condition ambient density $(\rho)_0$ and temperature $(\lambda)_0$ are defined. These quantities then define the ambient pressure as

$$(p)_0 = \frac{(\rho)_0}{2(\lambda)_0}. \tag{2.154}$$

Inflow is allowed, if the internal pressure is lower than the ambient pressure. The inflow density and temperature are set to ambient conditions. In the case that the internal pressure is higher than the ambient pressure, zero gradient for density and temperature are used. The density is

$$(\rho)_{-1} = \begin{cases} (\rho)_1 & \text{for} \quad (p)_1 > (p)_0 \\ (\rho)_0 & \text{else} \end{cases} \tag{2.155}$$

and similarly the temperature is

$$(\lambda)_{-1} = \begin{cases} (\lambda)_1 & \text{for} \quad (p)_1 > (p)_0 \\ (\lambda)_0 & \text{else.} \end{cases} \tag{2.156}$$

The velocity is limited by a maximal inlet velocity U_{max} in the inflow case and else set to zero, such that

$$(\vec{U})_{-1} = \begin{cases} \vec{0} & \text{for} \quad (p)_1 > (p)_0 \\ (\vec{U})_1 & \text{for} \quad (p)_1 < (p)_0 \quad \text{and} \quad |\vec{U}_1| < U_{max} \\ (\vec{U})_1 U_{max}/|\vec{U}_1| & \text{else.} \end{cases} \tag{2.157}$$

2.5.5 Periodic boundaries

For several synthetic flow problems it is beneficial to virtually extend the domain by applying periodic boundaries, such that one side of the domain is seamlessly connected to the opposing side. Here, the periodic boundaries are implemented based on ghost cells. The whole flow state is simply copied from the opposing side of the domain, i.e.

$$(\underline{W})_{-1} = (\underline{W})_N, \tag{2.158}$$

where the index N denotes the last cell on the opposing side of the domain.

2.5.6 Symmetry boundaries

A symmetry boundary condition is implemented by copying the flow state from the domain cell to the ghost cell, i.e.

$$(\underline{W})_{-1} = (\underline{W})_1. \tag{2.159}$$

In order to prevent in and out flow, the boundary condition inverts the velocity normal to the wall, e.g.

$$(U)_{-1} = -(U)_1 \tag{2.160}$$

if the boundary face is normal to the x-axis. Hence, this boundary condition acts as a slip boundary condition.

2.5.7 Creeping mass flux boundary

Finally, it is possible to implement Neumann type boundary conditions directly into the finite volume method. The fuel sources of the fire simulations later in this work are modeled by a creeping mass flux boundary condition, which is of Neumann type. The term creeping denotes that the mass brings no momentum into the domain but only mass, energy and amounts of passive scalars. The boundary mass flux is

$$\mathcal{F}^{\rho} = (U)_0(\rho)_0 \tag{2.161}$$

with $(U)_0$ being the small inflow velocity normal to the boundary. The corresponding energy flux is

$$\mathcal{F}^{\rho E} = \frac{K+3}{4(\lambda)_0}\mathcal{F}^{\rho} \tag{2.162}$$

and the passive scalar flux is

$$\mathcal{F}^{\rho\Theta} = (\Theta)_0\mathcal{F}^{\rho}. \tag{2.163}$$

2.6 LES turbulence modeling

In this work GKS is used under the framework of Large Eddy Simulation (LES) [103]. LES is based on a separation of scales, motivated from the energy cascade [104]. Turbulent kinetic energy is added to the flow at large scales, e.g. the scale of obstacles. Due to vortex stretching, energy is transferred in a cascading process to smaller scales [105, Chapter 5.1.2]. The very smallest scales carry near to no energy, but dissipate most of the energy. From this observations, LES is constructed to resolve the large scales and part of the cascade and model the dissipation of small scales. Hence, an LES model usually involves a filter, to filter out small scales, and an eddy viscosity model for the dissipation.

In this work the flow field is implicitly filtered by the finite grid resolution Δx. The eddy viscosity of the static Smagorinsky model [106] is implemented as given in [107, Chapter 4.2.8]. The turbulent viscosity is

$$\mu_t = \rho \left(C_S \Delta x\right)^2 S \tag{2.164}$$

with

$$\begin{aligned} S^2 = &2\left(\frac{\partial U}{\partial x}\right)^2 + 2\left(\frac{\partial V}{\partial y}\right)^2 + 2\left(\frac{\partial W}{\partial z}\right)^2 \\ &+ \left(\frac{\partial U}{\partial y} + \frac{\partial V}{\partial x}\right)^2 + \left(\frac{\partial U}{\partial z} + \frac{\partial W}{\partial x}\right)^2 + \left(\frac{\partial V}{\partial z} + \frac{\partial W}{\partial y}\right)^2 - \frac{2}{3}\left(\nabla \cdot \vec{U}\right)^2. \end{aligned} \tag{2.165}$$

Therein, $C_S = 0.2$ is the Smagorinsky constant as used in FDS [107, Chapter 4.2.1]. This value for C_S is valid for isotropic turbulence [102, Chapter 9.3.1]. The theoretical analysis of Lilly [108] reveals a value of $C_S = 1.7$ and the in the book of Blazek [101, Chapter 7.3.3] this analysis is cited with $C_S = 1.8$. This shows, that the value for C_S is by no means unique. Furthermore, it is known that in shear flows, the value must be reduced to $C_S = 0.1$ [101, Chapter 7.3.3] or even less [102, Chapter 9.3.1].

The physical viscosity is replaced by the turbulent viscosity when $\mu_t > \mu$. In this case, also the Prandtl number is changed to a turbulent Prandtl number of $Pr_t = 0.5$ and the diffusivity is modified based on the turbulent Schmidt number $Sc_t = 0.3$, such that $D_t = \mu_t / Sc_t$.

2.7 Grid refinement

Solving flow problems on uniform grids is considered inefficient in many cases. Flow fields usually show different degrees of complexity at different locations in the flow field. Therefore, it is imperative for a computational flow solver to allow for varying cell sizes over the flow domain. For arbitrary unstructured grids, this is solved by continuously reducing the cell size towards the areas of high complexity. On a uniform Cartesian grid this is not possible by construction. In order to allow different resolutions multiple grids with different resolutions are coupled. Here, an octree-type grid is used. This means that each coarse cell (parent cell) in the refinement region is split into eight smaller cells (child cells) with half the side length, as shown in Fig. 2.7. The different grids are termed refinement levels, where the level denotes the depth of a cell in the octree. Hence, the cells on the coarse grid are on level 0, the next finer cells on level 1, and so on.

For different flow solvers different approaches of grid refinement have been published. For LBM, for instance, methods with overlapping [109] and staggered grids [110] have been proposed and used in many applications, e.g. the simulation of urban wind flows by Lenz et al. [111]. In the finite volume world octree-type grid refinement is not common due to the possibility of continuous refinement on arbitrary unstructured grids. Some examples of octree based refinement can be found in the works of Olshanskii

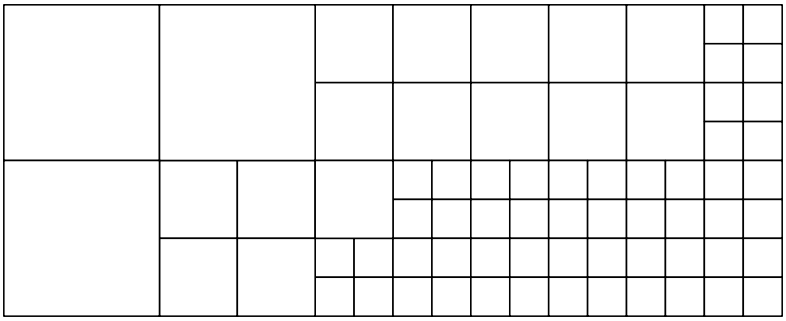


Figure 2.7: Two-dimensional non-uniform Cartesian grid with 3 grid levels and complex refinement region.

et al. [112], Pletzer et al. [113], Guittet et al. [114] and Batty [115]. The way the differing grids are coupled depends strongly on the solution method and cannot be transferred to other methods directly. Quadtree-type refinement for GKS has been shown by Yuan et al. [116]. In their reconstruction, variables from the finer grid are interpolated to obtain the coarse grid behavior. Unfortunately, not many details on the coupling are given in their paper. More details are provided by Lyu et al. [117] for their Cartesian grid solver for potential flow. The authors propose to compute gradients on cell faces on the interface between coarse and fine grids by a least-square approach. Due to the non-uniformity of the resulting stencil, second-order accurate gradients require large stencils that are different from one cell to another.

In the following sections an octree-type refinement for smooth flow GKS is presented. It was first published by Lenz et al. [97] for two-dimensional flows in 2019.

2.7.1 Nested time stepping

Before the numerical treatment of the interface is explained the concept of nested time stepping is introduced. The stability of explicit schemes is related to the CFL number [118]

$$CFL = \frac{|\vec{U}|\Delta t}{\Delta x} \tag{2.166}$$

that relates spatial and temporal resolution. A requirement for stability is usually $CFL < 1$. On an arbitrary unstructured grid Δx varies continuously over the domain. The global CFL number of a flow with uniform velocity, hence, depends on the smallest cell. To satisfy consistent time stepping, the time step must be the same everywhere. This is called synchronous time stepping. The refinement with a fixed factor of 2 (or any other integer refinement factor) allows what is called nested time stepping. Instead of satisfying a global CFL number, a local CFL number for each grid level is computed. In order to obtain the same CFL number on each grid level finer grids need to have smaller time steps. More accurately, the coarse grid has a time steps twice as long as the fine grid since the fine grid usually decides the time step length. Nested time stepping is only possible because two fine time steps make

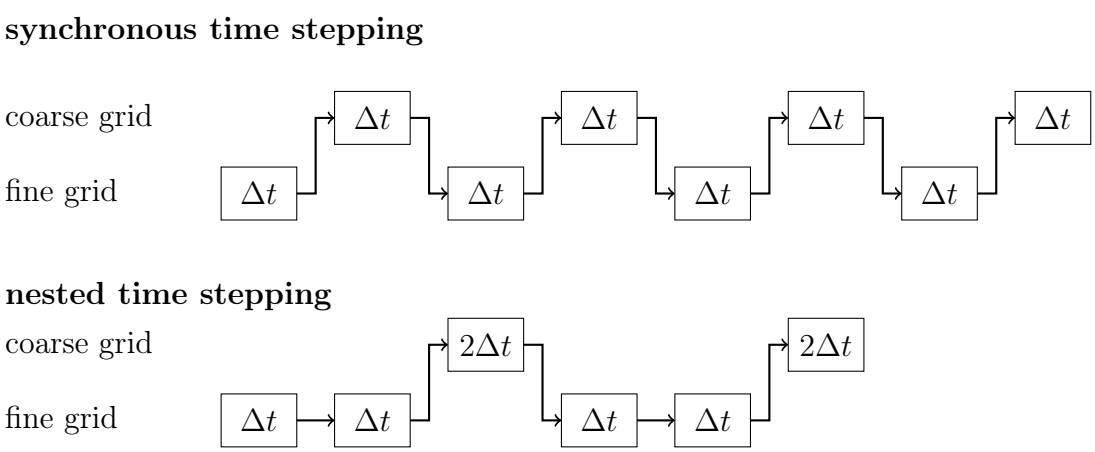


Figure 2.8: For synchronous time stepping the time step is equal on both coarse and fine grids. The grids are advanced in time alternately. For nested time stepping the coarse grid advance by twice the time in one step. This reduces the computational cost.

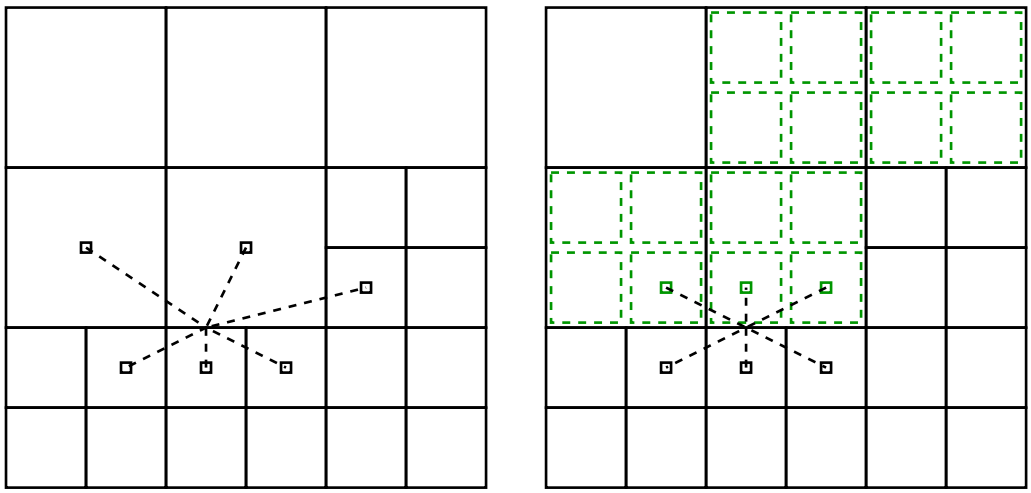


Figure 2.9: On cell faces on the interface, the stencils become arbitrary and non-centered (on the left). Therefore, ghost cells (on the right with dashed green lines) are introduced to allow the usage of the same stencils as on the uniform grid.

up one coarse time step, like two fine interfaces (in 2D) make up one coarse interface. The procedure of nested time stepping is shown in Fig. 2.8. Having to advance the coarse grid fewer times saves computation time and, hence, makes the simulation more efficient.

2.7.2 Interface ghost cells

The reconstruction stencil shown in Fig. 2.4 obtains its second order of accuracy from the property that it is centered, i.e. the point where the flow state and the gradients are computed lies in the exact center of the stencil. Due to this, the uneven error terms in the Taylor expansion vanish and the approximations are automatically second order accurate. Looking at the available stencils at a coarse to fine interface (see Fig. 2.9) reveals that these stencils will never be second-order accurate. Second-order reconstructions can be obtained by taking more cells into account. Such a stencils would look different for different interface topologies and, hence, complicate the computation.

In the staggered grid LBM by Geier et al. [110] coarse and fine grids overlap to allow interpolation from coarse to fine and fine to coarse. This idea is also used here, albeit in a different way, due to the difference that in LBM a compact interpolation with kinetic gradient information is possible. In order to allow the usage of the stencils in Eqs. (2.140) and (2.141) fine ghost cells are introduced all around the fine grid. Further, for every coarse cell on the interface all child cells (4 in 2D, 8 in 3D) are used (see Fig. 2.9). This allows for arbitrarily shaped refinement regions with the same algorithm and without any special cases.

2.7.3 Second order accurate ghost cell interpolation

Two interpolation rules for fine to coarse and coarse to fine are required.

Fine to coarse: The second-order interpolation of a coarse ghost cell is trivial because the coarse cell center lies in the middle of the fine child cells. The interpolated flow state in the coarse ghost cell is denoted by $\underline{\tilde{W}}_{0,0,0}$, where the tilde distinguishes the ghost cell from regular fluid cells. The indices denote the relative location of the cell. The corresponding child cells are denoted by $\underline{W}^{i,j,k}_{0,0,0}$ with $i,j,k \in \{-1,1\}$. The interpolation is

$$\underline{\tilde{W}}_{0,0,0} = \frac{1}{8} \sum_{i,j,k\in\{-1,1\}} \underline{W}^{i,j,k}_{0,0,0}. \tag{2.167}$$

Inserting Taylor expansions for $\underline{W}^{i,j,k}_{0,0,0}$ around $\underline{W}_{0,0,0}$, i.e.

$$\underline{\tilde{W}}_{0,0,0} = \underline{W}_{0,0,0} + \frac{\Delta x^2}{32}\left(\frac{\partial^2}{\partial x^2}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial y^2}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial z^2}\underline{W}_{0,0,0}\right) + \mathcal{O}(\Delta x^3), \tag{2.168}$$

reveals that this interpolation is indeed second-order accurate.

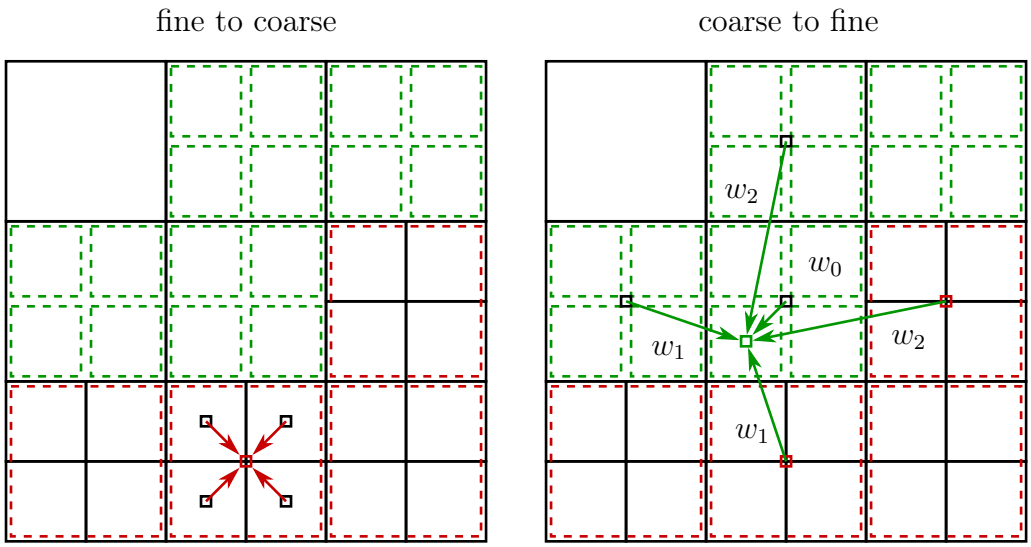


Figure 2.10: Ghost cell interpolation stencils: First, the value of the coarse ghost cell (dashed red) is computed from its fine child cells. Second, the value of the fine ghost cells (dashed green) are computed from their parent cells and the face neighbors of the parent cell.

Coarse to fine: In order to keep the second order of accuracy of the reconstruction and the GKS, the computation of the fine ghost cell values also has to be of at least second order. Due to the staggered arrangement of the cell centers resulting from the octree approach, fine cell centers are never in the center of a stencil of coarse cell centers. Therefore, second-order requires three points on the coarse grid for each coordinate direction. The obvious choice of coarse cells comprises the parent cell and the face neighbors, as seen in Fig. 2.10. Because the interpolation stencil extends over the interface, also a coarse ghost cell on the fine side of the interface is required to compute the fine ghost cells.

The interpolation stencil is formulated as

$$\underline{\tilde{W}}^{i,j,k}_{0,0,0} = w_0 \underline{W}_{0,0,0} + \sum_{\substack{l=i \ \vee \\ m=j \ \vee \\ n=k}} w_1 \underline{W}^{i,j,k}_{l,m,n} + \sum_{\substack{l=-i \ \vee \\ m=-j \ \vee \\ n=-k}} w_2 \underline{W}^{i,j,k}_{l,m,n}, \tag{2.169}$$

where w_0 is the weight of the parent cell, w_1 is the weight of the cells adjacent to child cell and w_2 the weight of the opposite cells, as seen in Fig. 2.10. These weights were chosen due to symmetry considerations. The indices l, m, n denote the relative location of the cells on the coarse grid. Inserting Taylor expansions for $\underline{W}^{i,j,k}_{l,m,n}$ and solving for the weights with the condition that the constant term yields $\underline{W}_{0,0,0}$ and the first-order term vanishes yields

$$w_1 = \frac{7}{24} - \frac{w_0}{6} \qquad \text{and} \qquad w_2 = \frac{1}{24} - \frac{w_0}{6}. \tag{2.170}$$

The solution leaves one weight open as there are two conditions and three weights. All choices for w_0 yield a second-order interpolation. The free weight can be set, such

that the interpolation becomes consistent in the sense that consecutive fine to coarse and coarse to fine interpolation yields the original cell state. Inserting the coarse to fine interpolation into the fine to coarse interpolation yields

$$\underline{W}_{0,0,0} = w_0 \underline{W}_{0,0,0} + \sum_{l,m,n \in -1,1} (w_1 + w_2) \underline{W}^{i,j,k}_{l,m,n}. \tag{2.171}$$

This equation is satisfied only for $w_0 = 1$. Hence, the interpolations weights are

$$w_0 = 1, \qquad w_1 = \frac{1}{8} \qquad \text{and} \qquad w_2 = -\frac{1}{8}. \tag{2.172}$$

It is remarkable that the weights are the same in one, two and three dimensions. Inserting Taylor expansions yields

$$\underline{\tilde{W}}_{0,0,0} = \underline{W}_{0,0,0} - \frac{\Delta x^2}{32} \left(\frac{\partial^2}{\partial x^2} \underline{W}_{0,0,0} + \frac{\partial^2}{\partial y^2} \underline{W}_{0,0,0} + \frac{\partial^2}{\partial z^2} \underline{W}_{0,0,0} \right. \tag{2.173}$$

$$\left. + 2 \frac{\partial^2}{\partial xy} \underline{W}_{0,0,0} + 2 \frac{\partial^2}{\partial xz} \underline{W}_{0,0,0} + 2 \frac{\partial^2}{\partial yz} \underline{W}_{0,0,0} \right) + \mathcal{O}(\Delta x^3). \tag{2.174}$$

This solution still contains six error terms of second-order. It is possible to choose the free weight w_0 in order to eliminate the second unidirectional derivatives, see Appendix B. This approach is not considered here, because the self consistency is not given.

2.7.4 Flux interpretation at the interface

The interpolations in the prior section allows the computation of ghost cell values, such that fluxes over the interface can be computed on both coarse and fine grids. It is not clear however, how the fluxes that are transported from one cell to another are interpreted in this context. Different options exist and will be introduced below.

- *Overlapped interface:* The obvious interpretation is that fluxes are computed on both coarse and fine grids. The ghost cells are included in the update and, hence, also advance in time before they are overwritten by the interpolations, which are only performed once per coarse time step. In the case of nested time stepping the fine ghost cells adapt to the change after one fine time step. A drawback of the *overlapped interface* is that the method is not conservative, as it cannot be assumed that the fluxes on the fine cell faces are exactly the flux on the coarse cell face.
- *Connected interface:* The defect of the *overlapped interface* can be corrected by only using the ghost cells for the reconstruction. The regular fluid cells are stitched together at the interface, such that fluxes are transported between coarse and fine cells directly. When considering nested time stepping this has

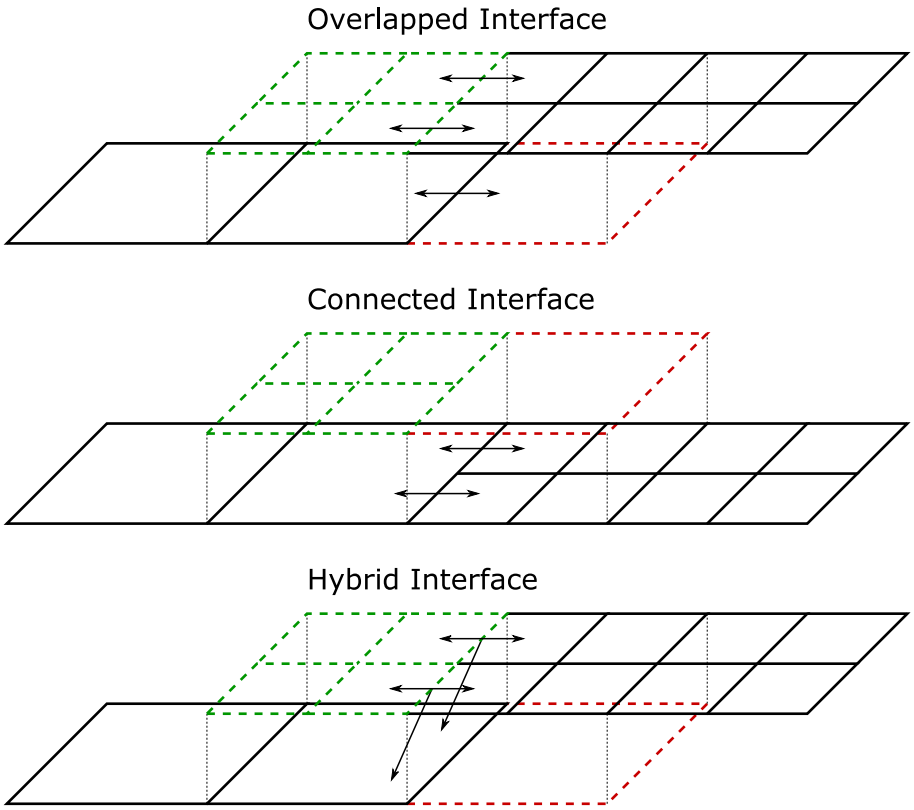


Figure 2.11: Three different flux interpretations at grid interface: The *overlapped interface* exchanges fluxes with the ghost cells, such that the interpolation is the only connection between the grids. For the *connected interface* fluxes are exchanged directly between coarse and fine fluid cells and the ghost cells are only used for the reconstruction. The *hybrid interface* combines the two interpretations. Figure reproduced from [97].

the disadvantage that after the first fine time step, the fine ghost cells have not adapted and keep their old value. The fine ghost cells act partially as a wall and reflect parts of waves passing through the interface.

- *Hybrid interface:* The *overlapped interface* and *connected interface* can be combined. The grids are stitched together to assure conservation and the fine ghost cells receive fluxes and are adapted after the first fine time step. This requires the flux towards the coarse side of the interface to be send to both the coarse fluid cell and the fine ghost cells.

The three different flux interpretations are shown in Fig. 2.11. The figure reveals an unclear point regarding the fluxes in the fine ghost cells. Not all fine ghost cells have all neighbors. Since no boundary conditions are defined at the boundary of the fine cells, the behavior of these cells is undefined and the cells get invalidated. It was stated earlier that always all fine child ghost cells are introduced. This corresponds to having two layers of fine ghost cells around the whole fine fluid region. The outer layer does not have all neighbors and, hence, is invalidated after the first fine time

step. This is acceptable, as these cells do not touch any fluid cells. In the second fine time step the inner layer of fine ghost cells gets invalidated as well, because fluxes between the inner and outer layer are exchanged. This is also acceptable, as both inner and outer layers are overwritten at the beginning of the next time step by the interpolation.

2.7.5 Validation of grid refinement

Three tests in two dimensions were performed to analyze behavior of the coarse to fine interface [97].

As a first test a pressure wave passes from a coarse region into a fine region and out again. The domain is practically one-dimensional and comprises $x = [0, 64L]$. The initial pressure wave is a Gaussian pulse centered at $x = 32L$ with an amplitude of 1% of the ambient pressure. Additionally, the background medium is advected with one percent of the speed of sound (i.e. $Ma = 0.01$). Two waves run through the domain, one in positive and one in negative direction. The grid is refined once in between $x = 47L$ and $x = 49L$, such that the wave traveling in positive direction passes the refinement, while the opposing wave does not. The solution after the wave passed the refined region is shown in Fig. 2.12. On the scale of the wave no reflections are visible. The reflections are about six orders of magnitude smaller than the wave, which by itself is a good result for all three flux interpretations. The smallest reflections are observed on the *hybrid interface* because it fixes the short comings of both of the other interpretations. In order to quantify the performance of the methods the L_2-errors of the amplitude in $x = [40L, 48L]$ and the maximal errors are investigated. For both metrics the *overlapped interface* and *hybrid interface* show second-order convergence for high resolutions, while the connected grid shows a lower convergence rate.

The second test case aims at investigating conservation and the effect of nested time stepping. Hence, a test case with a closed system and steady state solution is chosen. The lid driven cavity fulfils these criteria. This test was studied by many researchers and is, hence, regarded a standard test, even though it has shortcomings, such as inconsistent boundary conditions. The setup comprises a square cavity with three stationary walls and one moving wall (the lid). The Reynolds number for this setup was chosen to be $Re = 1000$ based on cavity width/height and lid velocity. The used grid has a background resolution of 64×64 cells and is refined twice towards the lid and the corners (see Fig. 2.13 (a)). The refinement has a complex shape. The velocity profiles match the common reference solution by Ghia et al. [119] well (see Fig. 2.13 (b)).

Results for both *overlapped interface* and *hybrid interface* are compared, where the latter was performed with both nested and synchronous time stepping. The *overlapped interface* disqualifies itself for future usage, as it is not conservative and, hence, does not converge. Fig. 2.14 (a) shows the residual change of the three simulations over wall clock time. The momentum in the *overlapped interface* does not converge. Fig. 2.14 (b) reveals that the simulation gains mass over time, which does not happen for the

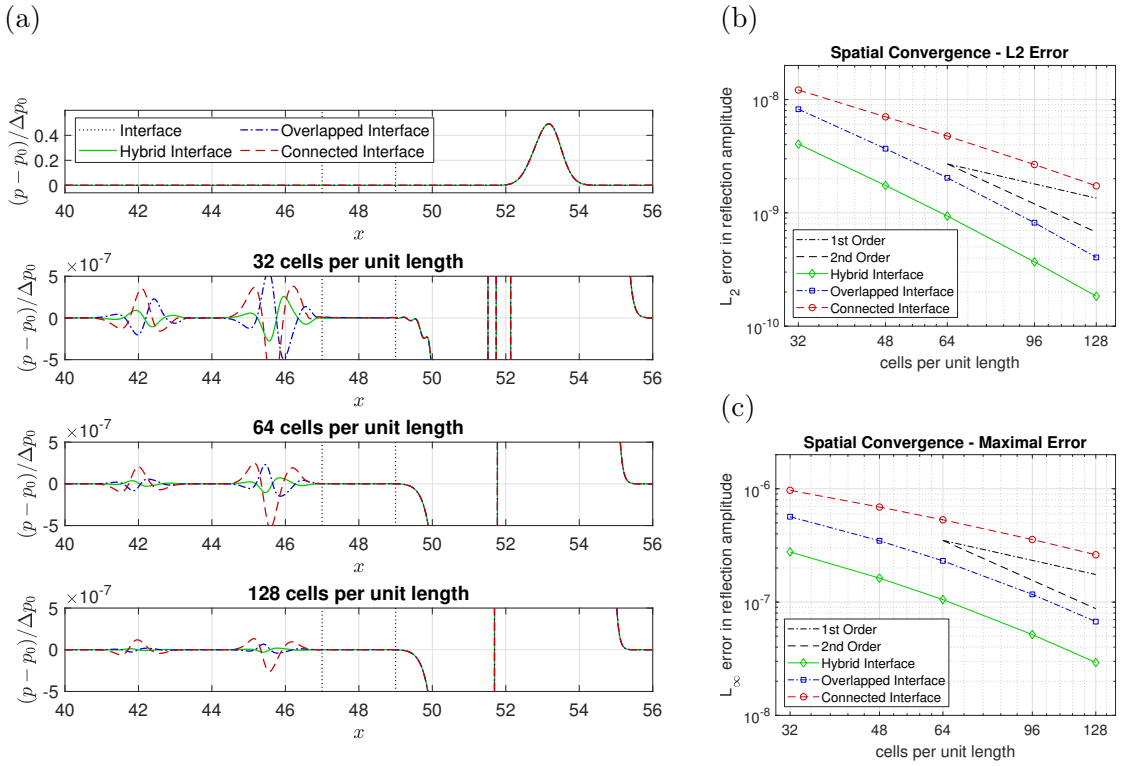


Figure 2.12: Results of the wave propagation test case with nested time stepping. A Gaussian pressure pulse is initialized at $x = 32$. The upper plot in sub-figure (a) shows the right going wave after it passes the interface. The lower plots show the same for different resolutions with a magnified pressure axis. The reflections are visible left of the refined region. The maximal amplitude of the reflection of the *hybrid interface* is nearly an order of magnitude smaller than the reflection of the *connected interface*. Convergence studies for amplitude of the reflected waves are shown in sub-figures (b) and (c). Sub-figure (b) shows the $L2$-error and sub-figure (c) the maximal error. The *hybrid interface* is second-order accurate. Figure reproduced from [97].

hybrid interface. This confirms that the *overlapped interface* is not conservative. In terms of time to solution, the nested time stepping is nearly twice as fast as the synchronous time stepping, see Fig. 2.14 (a). The three grids have 3798, 638 and 2216 cells from coarse to fine. The number of cell updates can easily be computed for both time stepping schemes. The nested time stepping requires 13,938 cell updates for one time step on the coarse grid. For synchronous time stepping the coarse grid has to perform four time steps (which cover less time), such that the required number of cell updates is 26,608. This shows, why the nested time stepping simulation converges faster per wall clock time by nearly a factor of two.

Finally, a two-dimensional channel flow is simulated with the *hybrid interface*. The channel has an aspect ratio of of $L/H = 32$. Both uniform and non-uniform simula-

(a) (b)

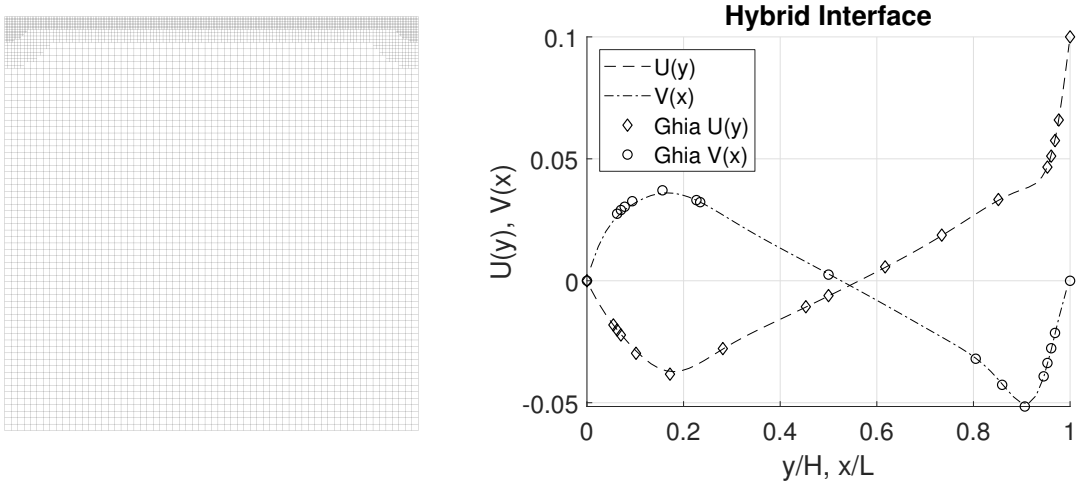


Figure 2.13: The grid for the lid driven cavity simulation is shown in sub-figure (a). The background grid has 64×64 cells and is refined towards the driven lid of the cavity twice. Sub-figure (b) shows velocity profiles along the center lines of the cavity at Reynolds number $Re = 1,000$. Our simulation compares well to reference [119]. Figure reproduced from [97].

(a) (b)

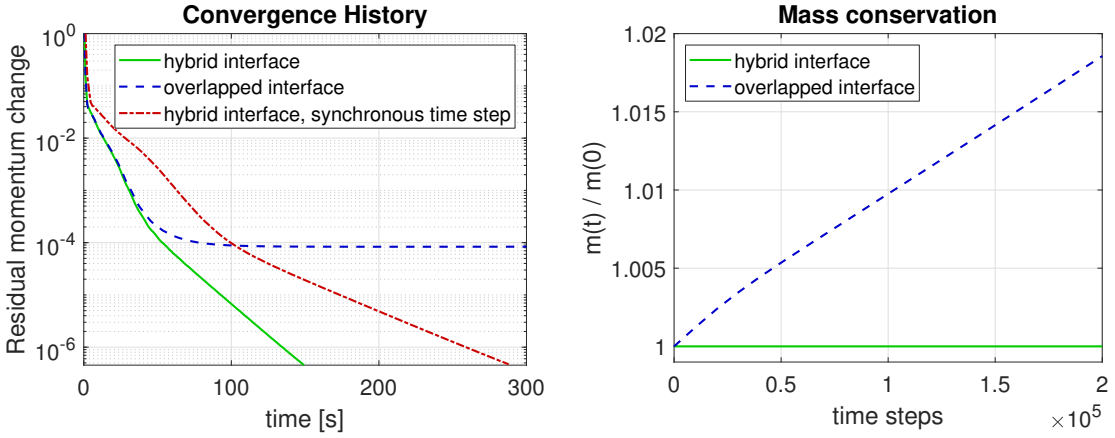


Figure 2.14: Sub-figure (a) shows the residual change of the flow field between two snapshots, 1,000 time steps apart. In sub-figure (b) the integral mass in the system over time is plotted. It is evident that the mass is conserved when using the *hybrid interface* and mass is gained when using the *overlapped interface*. Figure reproduced from [97].

tions were performed. The non-uniform grid is refined for $x/L > 1/2$. At the inlet a parabolic velocity profile with integral mass flux is prescribed. The outlet has a constant pressure boundary condition with pressure p_0. The analytic solution for the

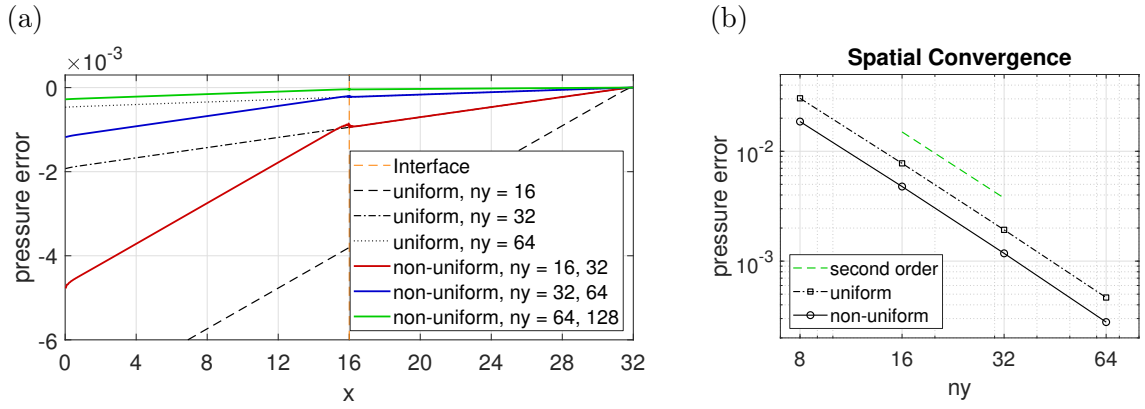


Figure 2.15: In sub-figure (a) the relative error in the pressure profile along a channel with fixed outlet pressure is shown. In the non-uniform simulations the second half of the domain is refined once. The non-uniform simulations show distinct kinks at the interface. It is evident that the increase in pressure error is only dependent on the local resolution. In the coarse and fine sections the slope of the pressure error is identical to the slope in the coarse and fine uniform simulations, respectively. Sub-figure (b) shows the relative error in the pressure at the inlet of the channel. The error converges with second-order. Figure reproduced from [97].

pressure p_a and for incompressible flow is known to be

$$p_a(x) = p_0 + g_x \rho (L - x), \qquad \text{where} \qquad g_x = 8\frac{U\nu}{H^2}. \tag{2.175}$$

In order to reduce compressibility errors the Mach number for this test is chosen as $Ma = 0.001$. The Reynolds number for this flow is only $Re = 10$, because the pressure outflow boundary condition is reflective and higher Reynolds numbers crash due to these reflections. The relative pressure error is computed as $(p(x) - p_a(x))/(g\rho L)$ (i.e. normalized by the pressure difference between inlet and outlet) and shown in Fig. 2.15 (a).

Going from the outlet (where the pressure is fixed) to the inlet, the pressure error increases linearly for the uniform simulations. The non-uniform simulations show a kink at the interface. The slope of the pressure error in both fine and coarse regions coincide with the respective uniform simulations. This is because the error in pressure loss is related to the local resolution. At the interface the pressure shows small wiggles. Comparing the inlet pressure error for different resolutions shows that the present method is second-order convergent in space.

3 Fire modeling and simulation

This chapter deals with the modeling of fire and its implementation in the GKS framework developed in this work.

3.1 What is fire?

Fire is an uncontrolled combustion that emits sufficient amounts of light and energy to be perceptible and self sustaining [120]. Fire is known to mankind for a long time, and for a million years mankind uses fire for different practical purposes [1]. The distinction between fire and combustion is that as combustion describes the underlying chemical process. The term combustion is usually used in technical systems, where a controlled chemical reaction is performed, often under optimized conditions. The term fire is usually used for natural self sustaining combustion. This ranges from voluntary bonfires, fireplaces and hearths to involuntary fires of buildings and wild fires.

Combustion is an exothermic reaction where a gaseous fuel oxidizes with oxygen (often as part of air). The rate of the chemical reaction depends on the temperature and the fractions of the two reactants. High temperatures typically imply fast reaction, whereas low temperatures usually lead to slower reaction dynamics. Chemical reactions posses an energy threshold, called activation energy, that has to be exceeded before the reaction starts. The energy released in the case of exothermal reactions implies that more energy is released than lost and the reaction becomes self sustaining.

Two modes of combustion can be distinguished by the mixing of fuel and oxygen. In technical applications premixed combustion is often desired. Igniting the (cold) mixture at one point leads to a premixed flame front that travels through the gas, separating into an unburned and burned part. The interface of these two is very thin when the reactions are fast. Premixed flames are for instance found in car engines, where the fuel is mixed with air before it is ignited. Point ignition takes place in Otto engines, while Diesel engines have a distributed ignition that is caused by pressure increase in the entire volume.

Non-premixed combustion is less efficient, because the reaction can only take place at points in space where both reactants are present and the temperature is above the activation energy. The reactants are initially separated, such that mixing has to occur, before the reaction can take place. On a molecular level mixing can only occur

in terms of molecular diffusion. Hence, such flames are called diffusion flames. In a resting fluid mixing is very slow, because the molecular diffusion of the species is low. Non-premixed combustion is often driven by turbulent mixing, which increases the local species gradients substantially to allow sufficient diffusion to mix the reactants.

Fire is the prime example for non-premixed combustion. A schematic for a burning solid fuel is shown in Fig. 3.1. The first step is the pyrolysis. Pyrolysis is a thermal decomposition process of a solid material that emits gases from that solid. If these gases are flammable, the solid can burn. During the ignition the solid is heated up, such that the pyrolysis emits the gaseous fuel. Due to the high temperature the fuel ignites and hence provides energy for both the pyrolysis and the combustion reaction. The hot gases rise due to natural convection and thereby suck fresh air into the flame. After the ignition the flame has a core that is nearly pure fuel. The outside region of the flame is the reaction zone. This is the only location where combustion can take place, because all fresh air that is mixed into the flame reacts as a matter of a short time. Such a reaction zone is usually small. The reaction is fast since the reaction region is usually the hottest part of the flame. The volume between the core and the reaction region is filled with a mixture of combustion products and fuel. The path of the gas flow is also shown in Fig. 3.1.

3.2 Combustion modeling for diffusion flames

Many different models exist to model combustion and fire. This section concentrates on diffusion flames.

3.2.1 Direct numerical simulation of combustion

The seemingly simplest way of simulating fire (and combustion in general) is the Direct Numerical Simulation (DNS). This means that the flow is modeled by the Navier-Stokes equations without any averaging, filtering or turbulence modeling. In the same way the chemistry is implemented directly without specific models for turbulent combustion. In practice, DNS is rarely ever used because it is only valid if all scales are well resolved. In practice the turbulent scales of the fluid motion can only be resolved for limited Reynolds numbers. In addition the spatial and especially temporal scales of the combustion have to be resolved.

If all scales are resolved, the chemical reactions can be implemented directly. Let

$$\nu_F F + \nu_{Ox} Ox \rightarrow \nu_P P \tag{3.1}$$

be a chemical reaction, where F is the gaseous fuel, Ox the oxidizer and P comprises the reaction products. The respective stoichiometric coefficients are ν_F, ν_{Ox} and ν_P. The reaction rate constant k of this reaction is given by [121, Chapter 12.11]

$$k = AT^{\alpha} \exp\left(-\frac{E_a}{R_u T}\right), \tag{3.2}$$

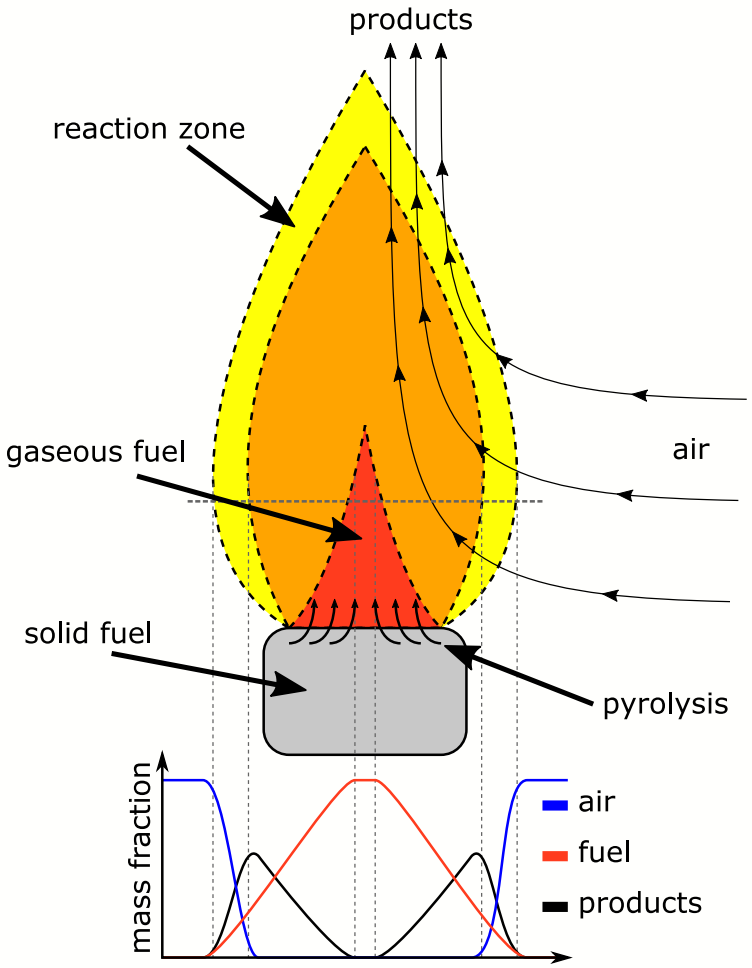


Figure 3.1: Schematic of a flame: The solid fuel is subject to pyrolysis that emits gaseous fuel into the flame. In the reaction zone gaseous fuel and oxygen from the air react to combustion products and release heat that drives the pyrolysis and the reaction, such that the fire is self sustaining.

where A is a pre-exponential constant, α is the temperature exponent and E_a is the activation energy. These three parameters are properties of the specific reaction. The reaction rate constant k is of the form of a modified Arrhenius equation (see [122, Chapter 2]). The temporal change of the species in the reaction are

$$\begin{aligned} \frac{d[Ox]}{dt} &= -\nu_{Ox} k[Ox]^a[F]^b, \\ \frac{d[F]}{dt} &= \nu_F k[Ox]^a[F]^b \quad \text{and} \\ \frac{d[P]}{dt} &= \nu_P k[Ox]^a[F]^b. \end{aligned} \tag{3.3}$$

Therein $[\cdot]$ denotes a concentration of a species and the exponents a and b are reaction parameters [121, Chapter 12.11]. The term $k[Ox]^a[F]^b$ is called reaction rate.

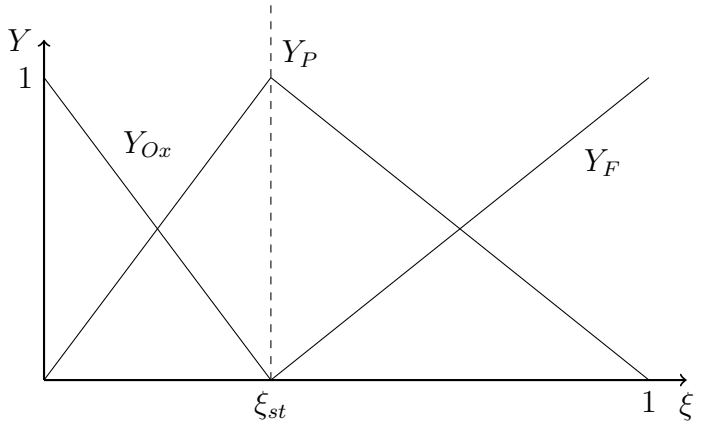


Figure 3.2: Relation between the species mass fractions Y and the mixture fraction ξ. Figure reproduced after [121, Chapter 12].

The ordinary differential equations for the species concentrations in Eq. (3.3) can be implemented directly into the solver. This is rarely ever done, because the time scales of the reaction are usually small compared to the feasible temporal resolution of the flow simulation. Further, the reaction zone is usually smaller than the feasible grid size. Hence, DNS for fire simulation is rare.

3.2.2 Simple chemical reacting system

The Simple Chemical Reacting System (SCRS) is one of the simplest ways to simulate laminar non-premixed combustion processes. The reaction is assumed to be a one-step reaction and it happens infinitely fast [121, Chapter 12.15]. The mixture of species is encoded in a single mixture fraction ξ that is 0 for pure fresh air and 1 for pure fuel. The third distinct value is the stoichiometric mixture fraction ξ_{st}. There the reactants are present in the exact stoichiometry of the reaction. The fractions of the species can be directly evaluated from the mixture fraction by linear relations, as seen in Fig. 3.2.

The mixture fraction is transported as a passive scalar that is subject to advection and diffusion. Additionally, if radiative heat transport is neglected, the enthalpy transport equation is equal to the transport equation for the mixture fraction, such that the temperature can also be evaluated from the mixture fraction [121, Capter 12.16]. This assumes that the mass diffusivity for all species and the thermal diffusivity are all the same.

3.2.3 Probability density function approach

The SCRS model in the prior section is only valid for laminar flows. Similarly to a RANS model, it can be extended to turbulent flows, but with Favre averaging instead of Reynolds averaging [121, Chapters 12.17 - 12.29]. Favre averaging accounts for

density fluctuations, which are important in combustion simulation. The velocity in RANS is decomposed as

$$U = \overline{U} + U', \tag{3.4}$$

where $\overline{u}$ is the average and u' are turbulent fluctuations. In Favre averaging, the velocity is decomposed into

$$U = \frac{\overline{\rho U}}{\overline{\rho}} + U''. \tag{3.5}$$

The fluctuation U'' accounts for velocity and density variations [121, Chapter 12.17]. This procedure reduces the number of unclosed terms in the averaged equations. The relation between the averaged mixture fraction and the mass fractions remains unclosed. Therefore, probability density functions (PDF) in ξ-space are introduced. The mass fractions are moments of the PDF with respect to the relations shown in Fig. 3.2. In order to efficiently solve the integration, β-PDFs are used to model the PDF [121, Chapter 12.20]. The coefficients in the β-PDF are computed from the Favre averaged mixture fraction and its fluctuation. The fluctuations are modeled by their own transport equation.

3.2.4 Eddy break-up model

In the eddy break-up model by Spalding [123] the fuel mass fraction is explicitly traced and, hence, is not required to be obtained from a PDF. The SCRS and the PDF approach both assume all fuel to instantaneously react on the stoichiometric iso surface. In the eddy break-up model, the rate of combustion is modified based on turbulent mixing. The turbulent mixing is quantified by the RANS turbulence model that predicts local values of turbulent kinetic energy and dissipation rate. Hence, the quality of the eddy break-up model depends on the quality of the turbulence model [121, Chapter 12.22]. It is possible to also use finite rate reactions based on reaction kinetics, i.e. the Arrhenius equation (see Section 3.2.1). In that case the lower one of the reaction rates predicted by direct kinetics and the turbulent mixing is used.

3.2.5 Laminar flamelet models

The above mentioned models all assume simple chemistry. More detailed chemistry can be incorporated by flamelet models. Therein, it is assumed that the local flame structure is one dimensional. The flame structure is read from a library of pre-computed laminar flamelets based on a local scalar dissipation rate. The pre-computed flamelets are much easier to solve and, hence, allow solution of many flamelets to gather data that makes up the library. The flamelets then recover the different species fractions and the temperature from the mixture fraction [121, Chapters 12.24 - 12.30].

3.2.6 Large eddy simulation for combustion

The modeling approaches in the prior sections can be implemented into RANS [102, Chapter 9.4] like flow solvers. That means that the flow solvers yield stationary solutions for the time averaged flow properties. The instationary nature of turbulent natural convection in flames is lost in this approach. The Large Eddy Simulation (LES) [102, Chapter 9.3] resolves the time dependent Navier-Stokes equations and models small scale turbulence (usually) by a turbulent viscosity. While requiring much more computational resources these methods provide a much better representation of turbulence, including the transient large scale structures. For modeling turbulent combustion and fire with LES, sub-grid combustion models are required. Even though, LES requires a much higher resolution than RANS simulations, the spatial discretization is usually much coarser than the flame thickness.

Two sub-grid combustion models were investigated by DesJardin and Frankel [124]. The conserved scalar approach utilizes a mixture fraction, flamelets and PDFs to reconstruct the flame structure. The direct closure approach directly traces multiple species with their own transport equations. The reaction in this approach is filtered in different ways. The reaction can be evaluated directly based on the filtered flow state but it can also be filtered itself. The sub-grid reaction is assumed to work similar to resolved scales, such that the sub-grid reaction is computed from the resolved scales. The reaction is evaluated based on Arrhenius equation. In this study, the direct closure approach with filtered reaction showed the best results.

The PDF method (with mixture fraction and flamelets) was applied by Sheikhi et al. [125] to a realistic premixed jet flame. A similar model for the simulation of a non-premixed swirl flame can be found in the work of Malalasekera et al. [126].

3.3 Fire simulation

The prior sections introduced combustion and fire. The presented modeling approaches were primarily designed for technical combustion occurring in specified volumes and with defined boundary conditions as opposed to involuntary fires. In the civil engineering community, which deals with fire protection, different models are employed. In this community fires in, on and around buildings (or compartments) are of concern. Quantities of interest are heat release rates, temperatures and emission of harmful substances, such as smoke, carbon monoxide and nitrogen oxides. These quantities depend (among other factors) on the material being burnt, as well as ventilation conditions of the compartment.

3.3.1 Zone models

A simple modeling approach for compartment fires are so called zone models. They are based on heat balances. The simple one-zone model traces the balance of heat

release by combustion (which is limited by ventilation), heat loss by outflow, heat loss by radiation through openings and heat flux into the walls for a single compartment [127, Chapter 10]. In the one-zone model, the compartment temperature is assumed to be the fire temperature. The wall heat flux is of interest because it affects the building materials in the walls. Together with the layout of the room and the heat release rate of the fire, the temperature development over time can be predicted. The one-zone model is considered for post-flashover fires. The flashover marks the change from limited fire of some of the fire load of a compartment to near instantaneous ignition of the whole fire load due to thermal radiation [128]. Due to the large amount of burning fire load, post-flashover fires are usually ventilation limited, such that the one-zone model applies.

Fuel limited fires can be modeled by the two-zone model. This applies before the flashover, where the fire is fuel limited, rather than ventilation limited. As a result, the compartment is partially filled with cold fresh air on the ground and hot burned air near the ceiling. To account for this, the compartment is split vertically into two zones [127, Chapter 11]. The upper zone has the flame temperature while the lower zone has ambient temperature. Mass flows from the lower zone into the upper zone in the fire plume. Fresh air enters the compartment in the lower zone and burned air leaves it from the upper zone. In the one-zone model only the flame temperature is unknown. In the two-zone model, also the height of lower zone is unknown. These two unknowns are computed from heat and mass balances, respectively. The heat release rate is bound by the maximal possible ventilation. Once this bound is reached, the two-zone model loses validity and a one-zone model can be used.

3.3.2 CFD for fire simulation

The zone models can be used for rough estimates. More detailed insight can be obtained by CFD simulation of the fire. In this section, the fire simulation CFD codes Fire Dynamics Simulator (FDS) and FireFOAM will be briefly introduced.

FDS (`https://pages.nist.gov/fds-smv/`) is the quasi standard for fire simulation. The software is developed primarily at the National Institute of Standards and Technology (NIST) in the United States. Several other research groups from all over the world have also contributed to the project. FDS targets the LES of low-speed compressible flows with emphasis on smoke and heat production in fires and their respective transport. It is applied on various scales varying from single compartments to whole buildings. The software package brings a great breadth of modeling options, material models and numerical choices. It models not only the fluid motion and combustion but also solid heat transfer, pyrolysis and solid degeneration, smoke production and transport, radiative heat transfer, fire suppression systems and finally evacuation in the FDS+Evac module. FDS brings a detailed documentation for both usage [5] and technical details [107]. FDS was verified [129] and validated [130] for hundreds of test cases. A drawback of FDS is that it does not scale well on modern hardware. High resolution FDS simulation requires many CPU hours. This is partly due to the fact that it does not scale well if spread to many CPUs.

The basic flow solver in FDS implements a finite volume method on uniform Cartesian grids. The low Mach-number assumption is used to get rid of acoustic modes. The information propagation speed is not limited by the speed of sound. This changes the mathematical properties of the governing Navier-Stokes equations drastically. The compressible Navier-Stokes equations are hyperbolic, meaning information travels only a finite distance in a finite time interval. The low-Mach number approximation introduces an elliptic pressure equation to the Navier-Stokes equation, similar as for incompressible flow. The elliptic pressure equation limits the scalability of such methods due to the required global operations. The pressure equation in FDS comprises many physical effects and is optimized for fire simulation. The temperature in FDS is extracted from the pressure equation and no separate equation is solved for temperature.

In terms of turbulence, FDS supports both LES and DNS (as DNS is usually a limit case of LES for very fine resolution). A wide range of different turbulence models such as static and dynamic Smagorinsky models, Deardorff's model, Vreman' model, the RNG model and the WALE model are implemented in FDS, while Deardorff's model is used by default. In terms of combustion modeling, FDS supports models over a wide range of complexity from simple mixture fraction models to direct simulations based on reaction kinetics and Arrhenius equation. Further, the user can specify arbitrary reaction equations for the combustion. In terms of turbulent combustion FDS implements a reaction time scale model. This model is an extension of the eddy dissipation combustion model by Magnussen and Hjertager [131], which itself bears similarities to the eddy break-up model in Section 3.2.4. Due to turbulence, cells may be partially mixed. The degree of mixture is extracted from different phenomena, depending on scale (i.e. grid resolution). The mixture processes are diffusion on very fine scales, advection on intermediate scales and buoyant mixing on larger scales. The minimum of the corresponding mixing times is used in the combustion [107, Chapter 5.2.1]. In the mixed part the chemical reaction takes place. If the reaction is much faster than the mixing, the "mixed is burnt" assumption holds. Hence, all mixed reactants are immediately reacted, such that no mixture of reactants remains. In this case, the procedure is similar to the eddy dissipation model [131]. Alternatively, Arrhenius equation can be used to compute the amounts of reacted reactants. FDS further comprises models for ignition and extinction.

Another software package for fire simulation is FireFOAM, an extension of the OpenFOAM software package, best known for its CFD solvers. FireFoam is similar to FDS in many regards. It also employs the eddy diffusivity model, albeit in a slightly different manner [6]. Moreover, FireFoam inherits the unstructured grid capabilities from OpenFoam, and is, hence, more versatile than FDS for complex geometries.

3.4 Fire model

This section introduces the fire model implemented in the present work.

3.4.1 Mixtures

Chemical reactions require a mixture of multiple species. In the gas kinetic scheme introduced in Section 2.3, only a single phase with density ρ is tracked. Additionally, passive scalar transport was introduced in Section 2.3.9. Passive scalars can be used to subdivide the single phase density into partial densities of multiple species.

The subdivision of the total amount of molecules can be done in two different ways. First, in terms of mass fraction Y and, second, in terms of mole fraction X. They are define respectively as

$$Y_i = \frac{m_i}{m} \quad \text{and} \quad X_i = \frac{n_i}{n}, \tag{3.6}$$

where m_i is the mass of the ith species and n_i is the amount of the ith species [121, Chapter 12.4]. The total mass and amount of substance are $m = \sum m_i$ and $n = \sum n_i$, respectively. The transformations between mass fraction and mole fraction are

$$X_i = \frac{M}{M_i} Y_i \quad \text{and} \quad Y_i = \frac{M_i}{M} X_i \tag{3.7}$$

with the molar mass of the mixture

$$M = \sum_i X_i M_i = \left(\sum_i \frac{Y_i}{M_i} \right)^{-1}. \tag{3.8}$$

Finally, partial densities can be obtained by $\rho_i = \rho Y_i$. The distinction between mass and mole fractions is important, as chemical reactions follow an integer stoichiometry, i.e. integer number of molecules react, justifying the use of mole fractions. The flow solver tracks mass as conserved variable and carrier of momentum, justifying the use of the mass fraction.

3.4.2 Combustion reaction

In this work an extension of the mixture fraction model is applied. The model is similar to the model used in the fifth version of FDS and is reported in the corresponding technical reference guide [132].

Here we consider only the one step combustion of methane with air:

$$CH_4 \; + \; 2O_2 \; + \; n_{N_2} N_2 \; \longrightarrow \; CO_2 \; + \; 2H_2O \; + \; n_{N_2} N_2 \; + \; \Delta H_{CH_4} \tag{3.9}$$

In this chemical equation the symbols for chemical species are to be interpreted as one mol of the species. The amount of nitrogen n_{N_2} accounts for nitrogen that accompanies the oxygen (which together make up fresh air) passively in the reaction. The reaction releases heat, quantified by the molar heat of combustion ΔH_{CH_4}, which is measured in terms of $\mathrm{kJ\,mol^{-1}}$. The mass specific counterpart of the molar heat of combustion is $\Delta h_{CH_4} = \Delta H_{CH_4} / M_{CH_4}$.

The chemical equation above shows that this simplified combustion happens in a mixture of 5 species. Instead of introducing five passive scalars for these five species, it is possible, to reduce the number of passive scalars. This reduces memory consumption and computation time. The methane (from now on called fuel) needs to be tracked as one passive scalar, as it can mix with both air and reaction products. The remainder of the left hand side of Eq. (3.9) is titled fresh air and contains fixed amounts of oxygen and nitrogen. The complete right hand side makes up the reaction products, which consist of fixed amounts of carbon dioxide, water and nitrogen.

The generalized species fuel, air and products are denoted by their mass and mole fractions Y_F, Y_A, Y_P, X_F, X_A and X_P, respectively. The sum of these three mass fractions must always be unit, i.e.

$$Y_F + Y_A + Y_P = 1 \quad \Leftrightarrow \quad Y_A = 1 - Y_F - Y_P. \tag{3.10}$$

This reduces the number of required passive scalars by one, such that finally two passive scalars are required.

The chemical equation (Eq. (3.9)) can be reformulated in terms of mole fractions, yielding

$$\Delta X_F \quad + \quad \Delta X_A \quad \longrightarrow \quad \Delta X_P \quad + \quad \Delta H_{CH_4}. \tag{3.11}$$

The stoichiometry of the original chemical equation can be used directly in the mole fraction equation, yielding $\Delta X_{O_2} = 2\Delta X_F$. Even though the generalized species are used, detailed knowledge of the oxygen content of air is required. For this reason the oxygen mole fraction $X_{O_2}^{\infty}$ in air is introduced by the relation $X_{O_2} = X_{O_2}^{\infty} X_A$. With this, the oxygen mass fraction is

$$Y_{O_2} = \underbrace{X_{O_2}^{\infty} \frac{M_{O_2}}{M_A}}_{Y_{O_2}^{\infty}} Y_A. \tag{3.12}$$

The task of the combustion model is to predict the amount of reacted fuel ΔX_F. For this, the "mixed is burned" assumption is used [107]. For under resolved fire simulations the flame thickness is much smaller than the cell size. Hence, a cell in the vicinity of the flame front will contain unburned fuel, fresh air, reacting fuel air mixture and products. In this way cells can be partially mixed. Further, the chemical reaction time scale is much smaller than the time step. This allows the assumption of instantaneous reaction in the mixed part of the cell. The mixing time scale in the FDS 5 combustion model [132] used here is

$$\tau = \frac{0.1\Delta x^2}{D_{LES}}, \tag{3.13}$$

where D_{LES} is the eddy diffusivity predicted by the sub-grid scale turbulence model. The mixed fraction of a cell is $\Delta t/\tau$.

For total instantaneous reaction the amount of burned fuel is determined by the lesser amount of fuel and oxygen in their stoichiometric mass ratio

$$s = \frac{\Delta Y_F}{\Delta Y_{O_2}} = \frac{\Delta X_F M_F / M}{2 \Delta X_F M_{O_2} / M} = \frac{M_F}{2 M_{O_2}}, \tag{3.14}$$

such that $\Delta Y_F = \min(Y_F, s Y_{O_2})$. Applying this only to the mixed part of a cell yields the volumetric heat release rate as

$$\dot{q}''' = \rho \frac{\min(Y_F, s Y_{O_2})}{\tau} \Delta h_{CH_4}. \tag{3.15}$$

The amounts of consumed fuel, oxygen and air over one time step are

$$\Delta(\rho Y_F) = \frac{\dot{q}'''}{\Delta h_{CH_4}} \Delta t, \tag{3.16}$$

$$\Delta(\rho Y_{O_2}) = \frac{\Delta(\rho Y_F)}{s} \tag{3.17}$$

and

$$\Delta(\rho Y_A) = \frac{\Delta(\rho Y_{O_2})}{Y_{O_2}^{\infty}} = \frac{\Delta(\rho Y_F)}{s X_{O_2}^{\infty}} = \frac{2 M_A}{X_{O_2}^{\infty} M_F} \Delta(\rho Y_F), \tag{3.18}$$

respectively. Finally, the update of the conserved variables comprises

$$(\rho Y_F)(t + \Delta t) = (\rho Y_F)(t) - \Delta(\rho Y_F) \tag{3.19}$$

$$(\rho Y_P)(t + \Delta t) = (\rho Y_P)(t) + \Delta(\rho Y_A) + \Delta(\rho Y_F) \tag{3.20}$$

$$(\rho E)(t + \Delta t) = (\rho E)(t) + \dot{q}''' \Delta t. \tag{3.21}$$

3.4.3 Quantification of the combustion model

The chemical reaction Eq. (3.9) considered in this work models the one step combustion of methane in air. The molar and mass specific heat release rates for methane are [121, Chapter 12.3]

$$\Delta H_{CH_4} = 800\,\mathrm{kJ\,mol^{-1}} \quad \Leftrightarrow \quad \Delta h_{CH_4} = 50000\,\mathrm{kJ\,kg^{-1}}, \tag{3.22}$$

respectively. The heat release rates can also be expressed in terms of consumed oxygen, which would make them independent of the specific fuel [133]

$$\Delta H_{O_2} \approx 468\,\mathrm{kJ\,mol^{-1}} \quad \Leftrightarrow \quad \Delta h_{O_2} \approx 13000\,\mathrm{kJ\,kg^{-1}}. \tag{3.23}$$

The oxygen content in air in this work is set in terms of mole fraction

$$X_{O_2}^{\infty} = 0.21 \quad \Leftrightarrow \quad Y_{O_2}^{\infty} \approx 0.233 \tag{3.24}$$

while the mass fraction Y_{O_2} is approximated.

3.4.4 Limiters

In order to stabilize the simulations and prevent non-physical results three limiters were implemented.

Temperature limiter: The temperature limiter should prevent excessive temperatures that appear at the front of fire plumes. For this purpose the thermal diffusivity is increased in presence of large temperature gradients

$$k_L = k + \min\left(k_{L,max}, C_T \Delta x^2 ||\nabla T||\right), \tag{3.25}$$

where k_L is the increased thermal diffusivity, $k_{L,max}$ is an upper bound and C_T is an adjustable constant.

Passive scalar limiter: A problem of the passive scalar transport is that due to dispersion errors, negative values might appear. These correspond to negative mass fractions and are, hence, non-physical. To reduce this effect the diffusivity in regions with negative or excessive passive scalars is increased. The new diffusivity is

$$D_L = \begin{cases} D - C_{PS}\Theta & \text{for} \quad \Theta < 0 \\ D + C_{PS}(\Theta - 1) & \text{for} \quad \Theta > 1 \\ D & \text{else,} \end{cases} \tag{3.26}$$

where D_L is the increased diffusivity and C_{PS} is a tunable constant for the passive scalar limiter.

Heat release limiter: On very coarse grids the heat release rate in Eq. (3.15) might be excessive and is hence limited to

$$\dot{q}'''_L = \max\left(\dot{q}''', \dot{q}'''_{max}\right), \tag{3.27}$$

where $\dot{q}'''_L$ and $\dot{q}'''_{max}$ and the limited heat release rate and a maximal allowed heat release rate. By computing the consumed reactants from the heat release rate, this limiter can easily be implemented in the reaction. Version 5 of FDS also features such a limiter [132, Chapter 6.1.4].

4 GPGPU implementation of the GKS

The gas kinetic scheme presented in the past chapters was implemented as part of the software package VirtualFluids. As of now VirtualFluids comprises three flow solvers and several branches off these solvers. Over the last fifteen years VirtualFluids was developed as an LBM solver first for distributed hardware [134, 135] and starting with the works of Tölke and Krafczyk [136] and Schönherr et al. [137] also for GPUs. Over time the two execution models diverged resulting in the separate solvers VirtualFluidsCPU and VirtualFluidsGPU. As a result of this work the third solver VirtualFluidsGKS is added to VirtualFluids. Due to employing different discretizations to the Boltzmann equation the code overlap is small at this point in time. VirtualFluidsGPU and VirtualFluidsGKS share a grid generator, because the grid layout for both GPU codes is similar.

Recently, a reduced version of VirtualFluids was made available to the public [20] and proposals for full deployment and unification are under way. This chapter focuses on the implementation of VirtualFluidsGKS, which is mostly independent of the other codes. Chapter 5 introduces the shared grid generator.

In this chapter C++ and CUDA key words are highlighted in blue and VirtualFluidsGKS classes are highlighted in red.

4.1 Parallelization models

Many engineering problems are too complex and too large to be solved on a single serial computer that executes one operation after the other. This problem can be overcome by executing many operations on different data concurrently, and thereby reducing the time to solution substantially. Apart from the time to solution, the shear mass of data needs to be stored and addressed. This is some times not possible in a single address space, such that distributed computing with interaction of many computers comes into play. In the following three sections the two major parallelization models, i.e. shared memory and distributed memory computing will be introduced.

4.1.1 Shared memory parallelization

Most modern computers from consumer products such as mobile devices to high performance computers have MIMD processors, where MIMD stands for Multiple Instruction Multiple Data. These processors have several cores that can work on independent tasks concurrently. On such architectures it is possible to split a large task between multiple cores. All cores can access the same memory, such that this procedure is called shared memory parallelization. Long loops of independent tasks can be split in as many parts as cores are available. This can result in a substantial speed up with few cores. A bottleneck for shared memory parallelization is the memory bandwidth. The single memory with its single address space can only process a limited number of data queries per time. This limits the bandwidth, i.e. the amount of data per time that is send from the memory to the processors and back. To reduce the effect of this bottleneck, Non-Uniform Memory Access (NUMA) architectures were introduced that have a single memory space, but direct access to certain memory hardware for each core. Nevertheless, the scaling of shared memory parallelization is limited.

4.1.2 Computing on GPUs - CUDA

A special case of shared memory parallelization was introduced at the outgoing twentieth century with the introduction of Graphics Processing Units, GPUs. GPUs were introduced to render pixels on a screen. The computation involved in this are usually algebraically simple, but the number of pixels states that have be computed frequently is large. Hence, from the beginning the task of GPUs was to handle many simple tasks, as opposed to CPUs that evolved to solve complex tasks. Hardware and software were originally specialized for pixel rendering including many hardware implemented tasks directly on the GPU. At the beginning of the twenty first century people abused GPUs to do other computations on them. This was a tedious task, because the problem had to be stated in terms pixel and vertex shaders and was termed as "hacking the GPU". In 2007, NVIDIA, a leading vendor of GPUs, introduced the Compute Unified Device Architecture (CUDA) [138]. From there on it was possible to write general purpose programs for NVIDIA GPUs. In 2009, the Open Computing Language (OpenCL) was introduced for platform independent GPU programming [139]. Performing non-graphic related computations on GPUs is called General Purpose compution on Graphics Processing Units (GPGPU). In this work only CUDA is considered, due to its ease of use and the availability of NVIDIA GPUs.

Fig. 4.1 shows an idealized comparison of CPU and GPU architectures. The CPU nowadays has few compute cores, called Arithmetic Logic Units (ALUs). In order to perform complex tasks these ALUs use quite a lot of die-space on the processor. Further, CPUs bring large control structures to allow switching between tasks on the limited number of cores. The die-space is then complemented by comparably large amounts of memory directly on the chip of the processor. This memory is used to cache data from the main CPU memory, which is off-chip and, hence, requires much longer to be read. Caching improves performance when the same data is used multiple

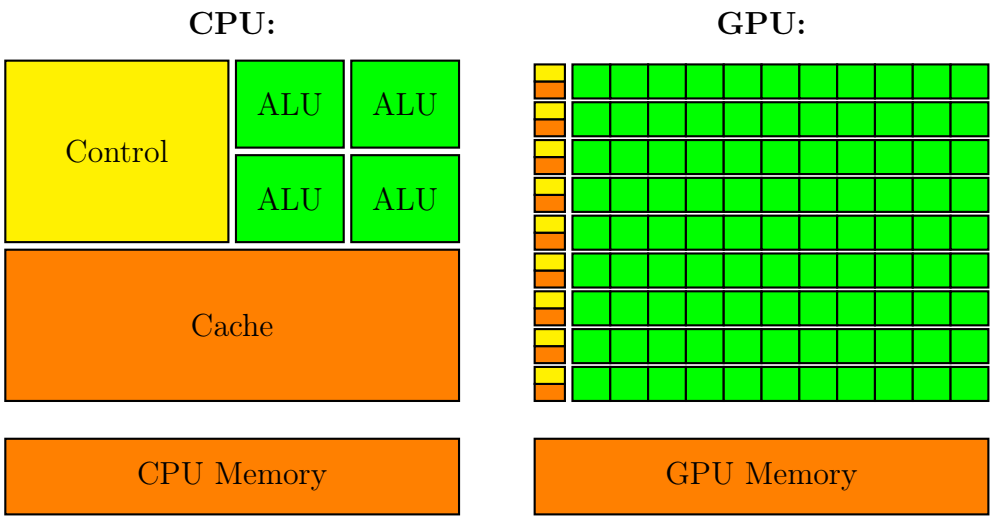


Figure 4.1: Comparison of CPU and GPU architectures, reproduced after [140].

times, or correlated data is used, where the control structures can predict future use of specific data.

The GPU has a shifted focus of die-space usage. In order to perform many parallel tasks, space is primarily used for ALUs. In order to save control structures, one control structure is used for multiple ALUs. This changes the architecture to SIMD, which stands for Single Instruction Multiple Data. This means, the same instruction is concurrently dispatched to several ALUs that perform this instruction for different data. One control structure with attached ALUs is called Streaming Machine (SM). Further, GPUs have only small caches. While early GPUs had no cache at all, modern GPUs bring substantial caches, as they are used more and more for general purpose tasks. Recent professional GPUs as the NVIDIA P100 bring 24 KB cache memory per SM and a GPU wide cache of 4 MB [141].

GPUs have dedicated memory on the graphics card, but off the GPU chip. This memory is connected with the maximal possible bandwidth in order to allow for high data throughput. In GPU computing the CPU side with the regular main memory is called host, since the regular systems hosts the graphics cards. The graphic cards are called devices. When writing a program that utilizes the GPU, the programmer has to explicitly manage device memory and copy data from host to device and vise versa. The GPU can only access data in the device memory[1]. In order to process the device data on the GPU, kernel functions have to be written.

An example kernel is given in Listing 4.1. It adds two vectors and stores the result in a third. The `__global__` qualifier signals that the function is to be called from the host and executed on the device. The pointer arguments must contain device memory addresses. The computation of `index` requires further explanation: SIMD implies that multiple data is processed simultaneously. For this purpose, CUDA divides the

[1] In modern CUDA unified access techniques allow the device to access host memory, which implicitly invokes copies and is not considered here for performance reasons.

Listing 4.1: Minimal CUDA code example

```
__global__ void kernel(float* A, float* B, float* C)
{
   int index = blockIdx.x * blockDim.x + threadIdx.x;
   C[index] = A[index] + B[index];
}

int main()
{
    float *A, *B, *C;
    cudaMalloc( &A, NUMBER_OF_BLOCKS * THREADS_PER_BLOCK * sizeof(float) );
    cudaMalloc( &B, NUMBER_OF_BLOCKS * THREADS_PER_BLOCK * sizeof(float) );
    cudaMalloc( &C, NUMBER_OF_BLOCKS * THREADS_PER_BLOCK * sizeof(float) );

    kernel <<< NUMBER_OF_BLOCKS , THREADS_PER_BLOCK >>> ( A, B, C );
}
```

large problem in blocks and threads, where a block contains multiple threads. A block is always assigned to one SM and threads of a block are executed simultaneously in chunks called warps, where the size of a warp is the number of cores per SM. All blocks have the same size, such that the index in the large problem can be computed by multiplying the block index `blockIdx.x` with the size of the block `blockDim.x` and adding the index of the thread in the block `threadIdx.x`. It is possible to use two- and three-dimensional blocks, but in this work only one-dimensional blocks are considered, hence the property `x` is used. The grid of blocks and threads is defined when dispatching the kernel as in the main function in Listing 4.1.

CUDA-C is a language extension for the C language (and also for C++). This extension defines a dispatch argument list signaled by `<<< ... >>>`. These dispatch arguments define the grid of blocks and threads. When a kernel is dispatched, the kernel is called `NUMBER_OF_BLOCKS * THREADS_PER_BLOCK` times, while the variables `blockIdx.x`, `blockDim.x` and `threadIdx.x` are set to the correct values for each thread. The device memory is managed by CUDA Runtime API calls, such as `cudaMalloc(void * dst, int size)`.

The execution order of the threads is completely up to CUDA. In fact, the SMs will continuously swap execution between multiple blocks in order to "keep the GPU busy". Memory access has a latency. Instead of idling, the SM will swap to another block (or other threads in the same block) as soon as one warp of threads is waiting for data. This procedure is called latency hiding. The fact that the execution order is arbitrary requires the threads to execute independently. CUDA brings synchronization routines on the block level, but these are not considered in this work, because the GKS as introduced above is easily separable into independent tasks.

GPUs usually are optimized for single precision computations, mainly because it is cheaper to implement and computation of pixel value does not require high precision. Professional GPUs such as the NVIDIA P100 bring double precision hardware for accurate computing, but they still have only half the number of double precision cores, compared to single precision cores. Fortunately, the direct simulation of turbulent flow in terms of LES or DNS does not require too much precision, as the turbulent flow is

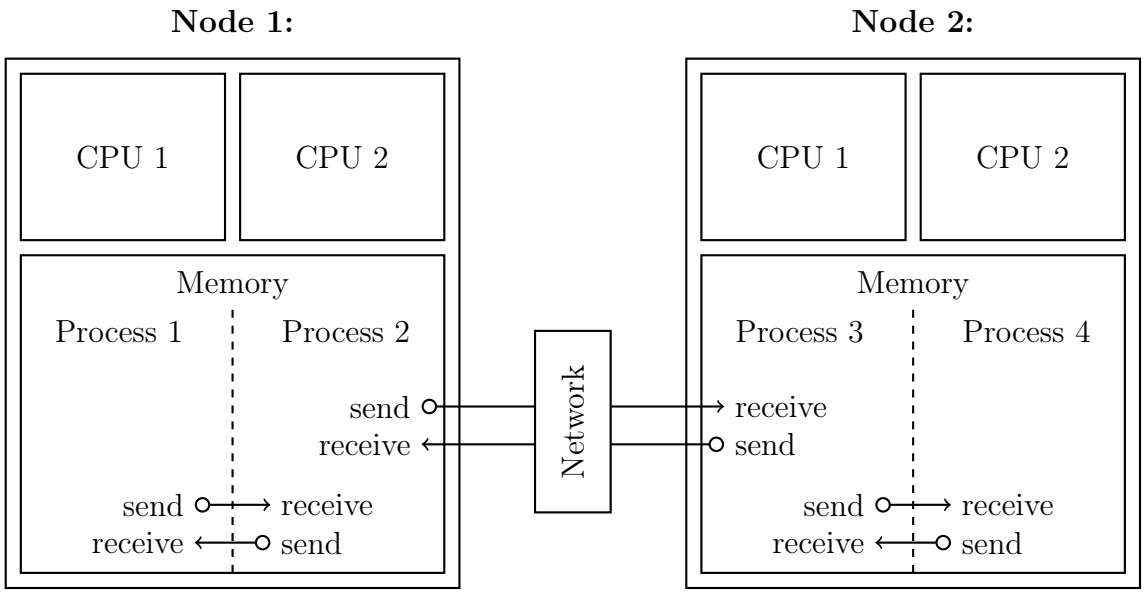


Figure 4.2: Distributed memory communication. Processes on the same node can communicate directly via the shared memory, while processes on different nodes have to communicate via the network.

probabilistic anyway.

Even though GPUs yield much higher performance than CPU shared memory parallelization, the same scaling barrier exists. In order to deal with larger problems multiple GPUs have to be utilized. The next section introduces distributed memory parallelization that can be used to utilize multiple GPUs.

4.1.3 Distributed memory parallelization

The prior two sections introduced shared memory parallelization on CPUs and GPUs and discussed their limited scaling. In order to overcome this scaling barrier, instead of building bigger computers, multiple computers are connected to form a cluster computer. All modern super computers are clusters of hundreds and thousands of independent computers called nodes. These nodes are connected by high speed networks, to exchange data. Running a computer program on such a cluster is called distributed memory parallelization, because each node has its own memory. The program is also executed in a distributed fashion. While a program on a single computer usually comprises a single process, a distributed program comprises multiple processes, distributed over the cluster. These processes explicitly or implicitly exchange messages to synchronize execution and exchange data.

The prevalent standard for explicit message passing in scientific computing is the Message Passing Interface (MPI) [142]. This standard has been implemented for various operating systems. MPI defines many different communication routines. In this work only point to point communication is considered.

Fig. 4.2 shows the explicit message passing in a cluster with two computers and four processes. In the figure it is assumed that each node has two cores (CPU 1 and CPU 2) and one process of the distributed program is run on each CPU, hence, having four processes. Each process owns its own memory in the node where it is executed. Point to point communication means that two processes take part in the communication. One provides the message data to send in a send buffer. The second process provides a receive buffer of similar length to receive the data. In order to exchange data, two communications need to be performed. MPI optimizes the communication, such that processes on the same node communicate directly via the system memory, while processes on different nodes need to communicate via the slower network. MPI follows the Single Program Multiple Data (SPMD) paradigm. This means every process executes the same program, but operates on different data, complying with the distributed memory parallelization. Similar to the CUDA threads that execute the same kernel, the MPI program knows what its part of the task is by its index, called rank.

MPI provides several point to point communication routines. Both send and receive routines exist in blocking and non-blocking forms. Blocking means that the execution of the code does not continue until the communication is complete on both sides. In order to exchange data between two processes, both have to call send and receive routines. Since there is no hierarchy between the two processes, the order of calling send and receive routines must be similar. Hence, blocking calls cannot be used, because calling a blocking send on both side would dead lock the program. Neither process could finish the communication, because the opposing process would have to call the receive routine, which it cannot because it is stuck in the send routine. For this reason non-blocking send (`MPI_Isend(...)`) and blocking receive (`MPI_Recv(...)`) are used in this work. Both processes call the send routine, which is then run by MPI in the background. Execution is returned to the program, such that the receive routine is called directly afterwards. This routine blocks until the whole message is received. Afterwards, the received data is used in the computation.

MPI is best utilized when the number of communications is small. For spatially resolved numerical simulations this is the case for schemes with local support. The domain can then be split into sub-domains and data only has to be exchanged at the boundaries of these sub-domains. This reduces not only the message length but also the number of communication partners per process. In terms of scaling, the amount of communication per process does not increase with number of processes. Schemes that have a global support, e.g. pressure based schemes in fluid dynamics, can be efficiently implemented using MPI for moderate number of cores. They will, however, never scale well to very large numbers of cores because the required communication per process increases with increased number of processes.

4.2 Code structure

In this section the basic aspects of VirtualFluidsGKS are discussed. It is implemented in C++ with both CUDA and MPI for hybrid distributed-shared memory computations. Further, the compute core can be executed on both CPU and GPU. First, this CPU/GPU execution dualism is introduced, followed by a detailed explanation of the data structures and the algorithms.

4.2.1 CPU/GPU dualism

The algorithms and data structures in VirtualFluidsGKS are clearly separated. The data and memory are managed by the `DataBase` class hierarchy that is shown on the left in Fig. 4.3. Objects of the `DataBase` class store a pointer `data` to the memory, which can be in CPU or GPU memory. The functionality of this class is implemented by a strategy pattern. This means the functionality is implemented in another class, which is known to the `DataBase` as a composition of the virtual type `DataBaseAllocator`. The implementation `DataBaseAllocatorCPU` allocates the memory in the host memory, while `DataBaseAllocatorGPU` allocates the memory in the device memory. By using the right allocator class, one can switch between CPU and GPU memory allocation. All other memory management functionalities, such as memory initialization, copies and freeing are implemented in these allocators as well. Finally, the allocators can be queried for their type to return either `"CPU"` or `"GPU"`.

On the algorithm side the dualism is implemented by splitting the computation in three functions, see Listing 4.2 for the example of the flux computation. The `FluxComputation::run()` function calls the computation from the host. The `fluxFunction` (generally referred to as `function`) implements the actual computation and accesses the data. The `fluxKernel` (generally referred to as `kernel`) is used only for GPU computation. The actual split between CPU and GPU is done by the template function `runKernel()` that is given in Listing 4.3.

The `deviceType` is passed to `runKernel()` as a string obtained from the `DataBaseAllocator`. Based on this string `runKernel()` either calls the function or iterates over the number of entities (i.e. the problem size) and calls the `fluxFunction` for each entity with the current index. The number of entities is stored in a `CudaGrid` struct together with the numbers of blocks and threads for GPU kernel execution. The template function `runKernel()` has a template argument `typename... TArgs` for a variable number of arguments. With this the specific arguments of the computation can be passed to both `kernel` and `function`, which themselves are passed to `runKernel()` as callable objects with template types. Additionally, the `kernel` gets the number of entities passed in order to perform the range check, see Listing 4.2. This is necessary, because the number of entities is not always a multiple of the number of threads per block. The `kernel` computes the `index` from the block and thread indices and calls the `function`, passing the `index`. In case of CPU execution, the `index` is directly passed to the `function`. Hence, `function` can be executed on

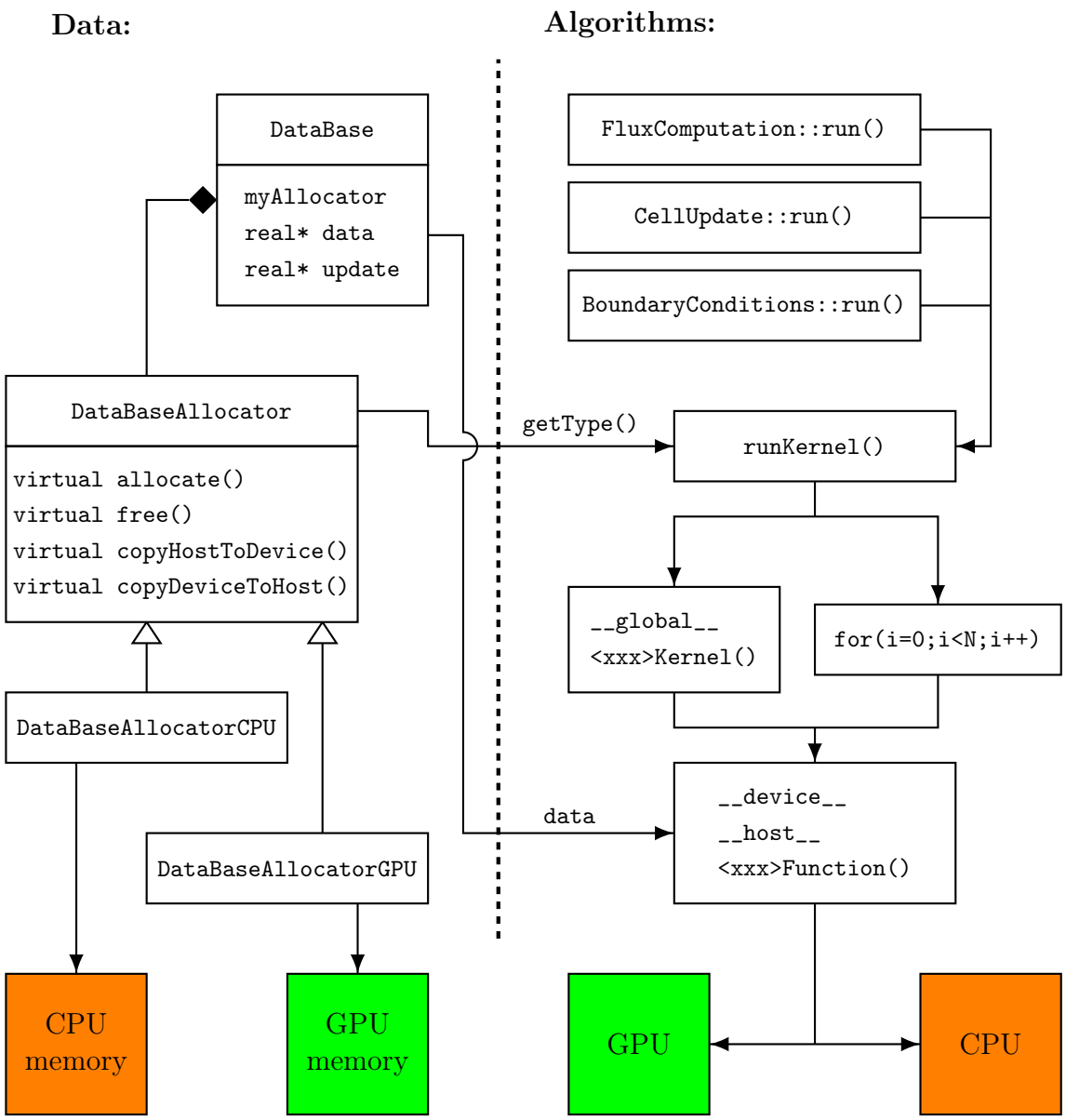


Figure 4.3: CPU/GPU dualism: On the data side a strategy pattern is used to manage memory on CPU or GPU memory. The virtual base class `DataBaseAllocator` and its implementations for CPU and GPU are used for all memory operations, such as allocation, freeing and copy operations. On the algorithm side the template function `runKernel` is used to split between CPU and GPU execution. Therefore, `runKernel()` queries the `DataBaseAllocator` for its type and based on that decides whether the kernel `<xxx>Kernel()` has to be launched or whether the iteration is performed on the CPU by simply looping over the number of entities `N`.

both CPU and GPU. Therefore, it must be compiled for both CPU and GPU. In the code this is signaled by the CUDA qualifiers `__host__` and `__device__`.

Listing 4.2: Required functions for the CPU/GPU dualism

```
__host__ __device__ inline void fluxFunction( DataBaseStruct dataBase,
                                              Parameters parameters,
                                              int index )
{
    [ ... computations ... ]
}

////////////////////////////////////////////////////////////////////////

__global__ void fluxKernel  ( DataBaseStruct dataBase,
                              Parameters parameters,
                              int numberOfEntities )
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;

    if( index >= numberOfEntities ) return;

    fluxFunction( dataBase, parameters, index );
}

////////////////////////////////////////////////////////////////////////

void FluxComputation::run(int level)
{
    CudaGrid grid(dataBase->perLevelCount[level].numberOfInnerFaces, 64);

    runKernel(fluxKernel,
              fluxFunction,
              dataBase->getDeviceType(), grid,
              dataBase->toStruct(),
              parameters
              dataBase->perLevelCount[level].startOfFaces);
}
```

Listing 4.3: Template function for CPU/GPU dualism

```
template<typename KernelFunctor, typename FunctionFunctor, typename... TArgs>
void runKernel(KernelFunctor kernel,
               FunctionFunctor function,
               string deviceType,
               CudaGrid& grid,
               TArgs... args)
{
    if( deviceType == "GPU" )
        kernel<<< grid.blocks, grid.threads >>>( args..., grid.numberOfEntities );
    else
        for( int index = 0; index < grid.numberOfEntities; index++ )
            function( args..., index );
}
```

4.2.2 Data structures

The explicit finite volume scheme in Chapter 2 can be implemented in different ways. The main data structures are discussed here.

The cells have a unique index over all levels. They are numbered, first by level, then by fluid cells before ghost cells and finally by location in the order z-y-x. The cell faces have a unique index as well and are sorted first by level and then by location. Further, the refinement interpolation cells have indices partitioned by coarse level,

Listing 4.4: Preprocessor macros that capsulate the memory layouts SOA and AOS

```
#ifdef SOA
    #define RHO__( cellIdx, numberOfCells ) ( 0 * numberOfCells   + cellIdx )
    #define RHO_U( cellIdx, numberOfCells ) ( 1 * numberOfCells   + cellIdx )
    [...]
#endif

#ifdef AOS
    #define RHO__( cellIdx, numberOfCells ) ( cellIdx * LENGTH_CELL_DATA     )
    #define RHO_U( cellIdx, numberOfCells ) ( cellIdx * LENGTH_CELL_DATA + 1 )
    [...]
#endif
```

because the interpolation is always initiated from the coarse level. In order to allow iteration over entities of one level, the following numbers are stored per level:

- number and start index of cells
- number and start index of faces
- number and start index of coarse to fine interpolation cells
- number and start index of fine to coarse interpolation cells

These indices are used to address the stored data.

For all arrays that store multiple data per entity the memory layout is capsulated by preprocessor macros to switch between Structure Of Arrays (SOA) and Array Of Structures (AOS) memory layouts, see Listing 4.4. By this, the memory layout can be switched easily. The flow state data $\underline{W}$ of all cells over all levels are stored in the single long array `DataBase::data`. A specific quantity in this array can be extracted with one of the macros, e.g. `RHO__` or `RHO_U`.

The finite volume update equation (see Eq. (2.136)) can be implemented in two different ways, which are called *Pull* and *Push* schemes in this work. In the *Pull* scheme the fluxes $\underline{\mathcal{F}}_j$ are stored per cell face during an iteration over the faces. During an iteration over the cells, the fluxes are read (i.e. *pulled* from the faces) and Eq. (2.136) is evaluated per cell. This procedure is absolutely thread-safe. In the *Push* scheme, the fluxes are accumulated per cell during the iteration over the faces (i.e. *pushed* from faces to cells). That means the flux over a cell face is added and subtracted to/from an accumulator variable of the cell on positive and negative side of the face, respectively. This procedure is not directly thread safe, because multiple threads can access the accumulator simultaneously, which can lead to read-modify-write race conditions. The problem can be resolved by using atomic operations that fuse the read-modify-write in a single operation. The memory footprint of the *Push* scheme is substantially smaller, because the number of faces is about three times the number of cells. Hence, storing the fluxes requires three times the memory than storing the accumulator. The *Push* scheme is implemented in VirtualFluidsGKS and the

accumulator DataBase::update is equal to

$$\sum_j A_j \int_0^{\Delta t} \underline{\mathcal{F}}_j dt, \tag{4.1}$$

see Eq. (2.136).

In addition to the flow state data, the unstructured connectivity has to be stored, since neighbor entities cannot be obtained from the indices alone. The required connectivities are:

- DataBase::cellToCell stores the indices of the six neighbor cells of all cells
- DataBase::faceToCell stores the indices of the cells on both sides of each face
- DataBase::faceOrientation stores whether the face is normal to the x, y or z direction. This is required for the transformation, see Section 2.4.3
- DataBase::fineToCoarse stores first the index of the coarse ghost cell on the interface and then the indices of the eight child cells
- DataBase::coarseToFine stores first the index of the coarse fluid cell on the interface and then the indices of the eight child ghost cells
- DataBase::parentCell stores the parent cell (in terms of octree refinement) of each cell. This is needed for the hybrid interface, see Section 2.7.4

Further connectivity can be obtained by pointer chasing. That means referencing first a direct neighbor where the connectivity is stored and then accessing the connectivity of the direct neighbor to address an indirect neighbor. This is for instance done for the computation of the tangential gradients, see Fig. 2.4, and for the coarse to fine interpolation, see Fig. 2.10.

Additionally, one accumulator DataBase::massFlux is required for the consistent source term treatment, see Eq. (2.120), and one accumulator DataBase::diffusivity for the diffusivity D_{LES}, see the mixing time scale in Eq. (3.13). Finally, an array of cell properties, stored as bitmap, is used to encode the cells ghost cell status (yes or no) and information about boundary fluxes.

4.2.3 Program flow

After introducing the data structures, the program flow (i.e. the algorithms) will be discussed in this section. The idea of nested time stepping was introduced in Section 2.7.1. Algorithm 1 shows how the nested time stepping is implemented in VirtualFluidsGKS. The algorithm is written in pseudo code and the symbol names differ from the ones in the code.

The recursive function nestedTimeStep expects a *level* and a time step length Δt as parameters. In the main loop of the simulation (i.e. the loop over all time steps),

Algorithm 1 Recursive nested time stepping for single GPU

```
1: function NESTEDTIMESTEP(level, Δt)
2:
3:     if level ≠ finestLevel then
4:         call FINETOCOARSEINTERPOLATION(level), Eq. (2.167)
5:     end if
6:
7:     call SETBOUNDARYCONDITIONS(level), see Section 2.5
8:
9:     if level ≠ finestLevel then
10:        call COARSETOFINEINTERPOLATION(level), Eq. (2.169)
11:
12:        call NESTEDTIMESTEP(level + 1, Δt/2)
13:        call NESTEDTIMESTEP(level + 1, Δt/2)
14:    end if
15:
16:    call COMPUTEFLUXES(level), Eq. (2.102)
17:
18:    call UPDATECELLS(level), Eq. (2.136)
19: end function
```

NESTEDTIMESTEP(0,$(\Delta t)_0$) is called, where $(\Delta t)_0$ is the time step on the coarsest grid, i.e. $level = 0$. The FINETOCOARSEINTERPOLATION is called from the current *level* and uses data from the next finer level, $level + 1$. Then the boundary ghost cell values are set. This has to be done before computing the COARSETOFINEINTERPOLATION, which follows. The reason is that the COARSETOFINEINTERPOLATION stencil may include boundary ghost cells. A this time, all interface ghost cells are computed and the next finer level can be evaluated. The NESTEDTIMESTEP function calls itself two times as NESTEDTIMESTEP($level + 1$,$\Delta t/2$). This is the heart of the nested time stepping, because the finer levels are evaluated twice with half the time step length. It is worth noting that the evaluations of the finer levels contribute fluxes to the coarse level due to the hybrid interface, see Section 2.7.4. When the finer levels are evaluated the fluxes on the coarse grid are computed by COMPUTEFLUXES. This could have also been done before the evaluation of the finer levels. Finally, UPDATECELLS computes the flow state in the cells in the new time step. This has to be done last in order to take all fluxes from the same level as well as the next finer level into account. This procedure is repeated for each time step.

The different phases in the nested time step algorithm are implemented following the procedure from Listing 4.2, i.e.

- `FluxComputation::run()` with `fluxFunction()` and `fluxKernel()`
- `CellUpdate::run()` with `cellUpdateFunction()` and `cellUpdateKernel()`
- `Interface::runCoarseToFine()` with `coarseToFineFunction()` and `coarseToFineKernel()`

Listing 4.5: Definition of boundary conditions in VirtualFluidsGKS

```
SPtr<BoundaryCondition> bcMX = std::make_shared<IsothermalWall>( dataBase,
                                                                 Vec3(0.0, 0.0, 0.0),
                                                                 lambdaHot,
                                                                 false );
SPtr<BoundaryCondition> bcPX = std::make_shared<IsothermalWall>( dataBase,
                                                                 Vec3(0.0, 0.0, 0.0),
                                                                 lambdaCold,
                                                                 false );

bcMX->findBoundaryCells( meshAdapter, true, [&](Vec3 center)
{
    return center.x < -0.5*L;
} );
bcPX->findBoundaryCells( meshAdapter, true, [&](Vec3 center)
{
    return center.x >  0.5*L;
} );
```

- `Interface::runFineToCoarse()` with `fineToCoarseFunction()` and `fineToCoarseKernel()`
- `BoundaryCondition::runBoundaryConditionKernel()`.

The classes `FluxComputation`, `CellUpdate` and `Interface` follow a clear separation of data and algorithms, since they only implement the computations, while the data is managed by the `DataBase`. This is possible, because this part of the computation is practically the same for all simulations. The Boundary conditions differ from simulation to simulation. Their implementation is discussed in the next section.

4.2.4 Boundary conditions

Because not every simulation uses the same boundary conditions, a polymorphic class hierarchy with virtual base class `BoundaryCondition` is implemented. All the actual boundary conditions introduced in Section 2.5 are derived from this base class and the method `BoundaryCondition::runBoundaryConditionKernel()` is overloaded. Further, each boundary condition implements its own `function` and `kernel` to comply with the CPU/GPU dualism.

The `BoundaryCondition` class stores pointers for lists with indices of ghost cells, fluid cells and second fluid cells. These lists are managed by the `DataBaseAllocator`, while the pointers are stored by the `BoundaryCondition`. They are generated by `BoundaryCondition::findBoundaryCells()`, which uses C++11 lambda functions to define the boundary region, where the current boundary condition should be used. Listing 4.5 shows an example how boundary conditions are defined. The third argument of `BoundaryCondition::findBoundaryCells()` expects an object of type `std::function<bool(Vec3)>`, i.e. a function object. In the example this function object is passed as a lambda function. The function object is called with the cell center of a ghost cell. It shall return `true`, if the ghost cell belongs to the boundary and `false` if it does not.

4.2.5 Pre-processing

In terms of pre-processing VirtualFluidsGKS depends on the grid generator that will be introduced in Chapter 5. Because the grid generator was designed initially for LBM simulations, an adapter was implemented that puts the data structures in the correct form, such that they can be copied by DataBaseAllocators for the simulation. The adapter uses high level data structures with classes for cells and faces internally to simplify the grid conversion. For the simulation, the low level data structures introduced in Section 4.2.2 are extracted from the adapter by the DataBaseAllocators.

The main tasks of the adapter are

- generate cell and face connectivity (DataBase::cellToCell, DataBase::faceToCell, DataBase::parentCell)
- sort and partition cells and face
- generate interface connectivity (DataBase::fineToCoarse, DataBase::coarseToFine)
- generate lists of possible cell pairs for periodic boundary conditions.

4.2.6 Post-processing

For the post-processing VirtualFluidsGKS uses the VTK library [143]. The adapter between VirtualFluidsGKS and VTK provides methods for generating VTK data structures from the VirtualFluidsGKS data structures. Further methods for writing VTK files of the generated VTK data structures, which can be post-processed in third party software such as Paraview [144], are provided. The VTK data structures can also be used for direct post-processing. Additionally, online analysis of the simulation can be done by analyzers introduced in the following section.

4.2.7 Analyzer

Several analyzers for online analysis are implemented into VirtualFluidsGKS:

- The CupsAnalyzer measures the computational performance of the simulation, cf. Section 4.3.
- The ConvergenceAnalyzer measures the change of the flow field between snapshots. This can be used to determine if a simulation is converged to a steady state.
- The EnstrophyAnalyzer and the KineticEnergyAnalyzer measure integral quantities for the validation on the 3D Taylor-Green vortex decay, cf. Section 6.2.1.

- The `PointTimeSeriesAnalyzer` records the time series of flow data at one point in the domain based on nearest neighbor interpolation.
- Multiple `PointTimeSeriesAnalyzer`s can be organized in a `PointTimeSeriesCollector`.
- The `TurbulenceAnalyzer` computes turbulent statistics based on Reynolds averaging.

4.3 Performance

The computational performance of a simulation software can be measured in multiple ways. One metric is the time to solution. This metric makes the computational performance of completely different methods comparable. This metric will be applied in Sections 6.2.3 and 6.2.4 to compare VirtualFluidsGKS to FDS.

In the LBM community the update rate is a typical metric for computational performance. It tells how many lattice nodes can be updated per second and is measured in Node Updates Per Second (NUPS). For GKS, this metric is adapted to Cell Updates Per Second (CUPS) and is computed as

$$\text{CUPS} = \frac{n_{\Delta t} \sum_{i=0}^{n_{levels}-1} n_{cells,i}\, 2^i}{\Delta t_{wall}}, \tag{4.2}$$

where $n_{\Delta t}$ is the number of time steps for the coarsest level, $n_{cells,i}$ is the number of cells for the ith refinement level and Δt_{wall} is the elapsed wall clock time.

In order to measure the performance, a simple test case on a square domain with all periodic boundaries is simulated. The simulation is run for 1000 time steps. Simulation of pure flow, without passive scalars and with forcing scheme (1), see Section 2.3.8, is compared to the full fire simulation with passive scalars and forcing scheme (3). The performance values are listed in Table 4.1. On a single CPU core on a usual desktop PC, the code has a performance of about 500,000 CUPS for pure fluid simulation. It is possible to utilize shared memory parallelization based on OpenMP for the loop in the CPU branch in `runKernel()`, see Listing 4.3. Utilizing four cores plus hyper-threading increases the performance by about a factor of five. On GPUs the code is about a hundred times faster, than on the CPU. For the consumer graphics card NVIDIA GeForce GTX 1080 Ti, which has 3584 cores and a theoretical memory bandwidth of 484 GB/s, the performance is about 200 MCUPS. On the professional NVIDIA Tesla P100 with the same number of cores and 732 GB/s a performance of about 300 MCPUS is measured. For double precision on the P100 the performance is reduced to about 175 MCUPS. The performance values for the simulation of fire are about 75 % of the pure flow simulations. This is due to two additional passive scalars that have to be traced.

Table 4.1: Computational performance of VirtualFluidsGKS on a single GPU

	flow only		including fire	
Hardware	MCUPS	BW (GB/ s)	MCUPS	BW (GB/ s)
Core i7-4770, 1 Core	0.5	0.5	0.47	0.6
Core i7-4770, 4 Core	2.5	2.6	2.2	2.8
GeForce GTX 1080 Ti, single	200	207.6	150	189.9
Tesla P100, single	300	311.4	215	272.8
Tesla P100, double	175	337.1	130	310.1

A challenge for parallel shared memory programs is getting enough data from the memory to the cores. To overcome this bottleneck GPUs do a great effort to achieve high bandwidths. The bandwidth utilization is another performance measure. For the computation of the utilized bandwidth, first the amount of memory that is read from memory per cell update needs to be known. Therein, the flux computation per face is counted thrice, because each cell has six cell faces, and each cell face is shared by two cells. For the pure fluid simulation in single precision that is

$$\underbrace{3}_{\text{3 faces per cell}} \cdot\Big(\underbrace{10\cdot 5\cdot 4\,\mathrm{B}}_{\text{read cell data}} + \underbrace{12\cdot 4\,\mathrm{B}}_{\text{read connectivity}} + \underbrace{3\cdot 1\,\mathrm{B}}_{\text{read properties}} + \underbrace{2\cdot 5\cdot 4\,\mathrm{B}}_{\text{write cell update}} + \underbrace{2\cdot 3\cdot 4\,\mathrm{B}}_{\text{write mass flux}}\Big)$$
$$+ \underbrace{5\cdot 4\,\mathrm{B}}_{\text{read cell data}} + \underbrace{5\cdot 4\,\mathrm{B}}_{\text{read cell update}} + \underbrace{3\cdot 4\,\mathrm{B}}_{\text{read mass flux}} + \underbrace{5\cdot 4\,\mathrm{B}}_{\text{write cell data}} = 1041\,\mathrm{B}. \tag{4.3}$$

In a similar fashion one cell update in double precision requires 1929 B. Including the passive scalars for fire increases the memory demand per cell to 1269 B and 2385 B for single and double precision, respectively. The utilized bandwidth BW for flow simulation in single precision on the P100 is

$$\mathrm{BW} = 300\cdot 10^6\ \frac{\text{Cell Update}}{\mathrm{s}} \cdot \frac{1041\,\mathrm{B}}{\text{Cell Update}} = 311.4\,\mathrm{GB/s}. \tag{4.4}$$

The utilized bandwidths for the different devices can be found in Table 4.1. It is observed that the utilized bandwidth, same as the update rate, scales with the bandwidth of the devices and not with the number of cores. This because the GTX 1080 Ti and the P100 differ mostly with regard to the bandwidth. The GTX 1080 Ti has about two thirds of the bandwidth of the P100 and achieves about two thirds of the utilized bandwidth.

The measured bandwidths are substantially smaller than the theoretical bandwidths, though. This is due to the fact that the theoretical bandwidth is computed based on hardware implementation, i.e. bus width times frequency. The maximal sustained bandwidth for the P100 was measured with the GPU-STREAM tool [145]. It performs a scaled vector addition of type $c_i = a_i + \alpha b_i$, where a_i and b_i are very long vectors and α is a scalar. The maximal sustained bandwidth of the P100 was measured as 543 GB/ s, which is only about 74 % of the theoretical bandwidth. Hence, the present implementation achieves about 57 % of the maximal sustained bandwidth. This is

still a rather small value compared with the bandwidth utilization of about 80 % that was measured for the two-dimensional GKS code [97]. The reason for this is related to latency hiding. GPUs frequently swap out tasks that are waiting for memory and continue other tasks instead. The granularity of the tasks are warps of 32 threads. The SMs of the P100 can hold up to 64 warps and have 65536 registers. In practice, the number of warps per SM depends on different factors. The limiting factor here is the number of registers required for the kernel, because the registers must remain untouched even if the warp is not processed at that time. The `fluxKernel` of the present three-dimensional code requires 138 registers, while the two-dimensional code only required 70 registers. Hence, the maximal number of warps per SM in 3D is limited to 14, giving an occupancy of only 22 %, while the occupancy of the 2D code is twice as much, i.e. 44 %[2]. The reason for the large number of registers required is that several expansion coefficients, moments, flow states and other variables must be hold simultaneously for evaluating Eq. (2.102). Having a low occupancy results in less efficient latency hiding, and, hence, not full utilization of the maximal sustained bandwidth.

4.4 Multi-GPU

As described in Section 4.1.2, the scaling of single GPU computing is limited due to memory bottlenecks and size limitations of the available hardware. This scaling barrier can be overcome by using more than one GPU to solve large problems. In order to solve the problem together the GPUs have to communicate. In this work the communication is implemented via MPI.

Before the implementation of the communication is described, two performance measures for distributed memory parallelization are introduced. A distributed program is usually slower than two independent instances of the same program. The reason is that the communication requires time, which is not needed by the independent program. The parallel performance can be measured in terms of speedup S and parallel efficiency E, which are introduced below. Starting from an independent program two ways exist to distribute the workload of the program, i.e. to scale the problem to more hardware. In the case of strong scaling (SS), the total problem size is kept constant with the aim to accelerate the solution of this problem. The problem size per process decreases and the ratio of communication to computation increases. In the case of weak scaling (WS), the problem size per process is kept constant with the aim to compute larger problems in the same time. Here, the ratio of communication to computation remains constant. The speedup is computed as

$$S_{SS}(N) = \frac{t(1)}{t(N)} \qquad \text{and} \qquad S_{WS}(N) = \frac{N \cdot t(1)}{t(N)} \tag{4.5}$$

[2]Occupancies are the ratio of warps per SM and maximal warps per SM. They can be computed by hand or with the CUDA Occupancy Calculator provided by NVIDIA, `https://docs.nvidia.com/cuda/cuda-occupancy-calculator/index.html`

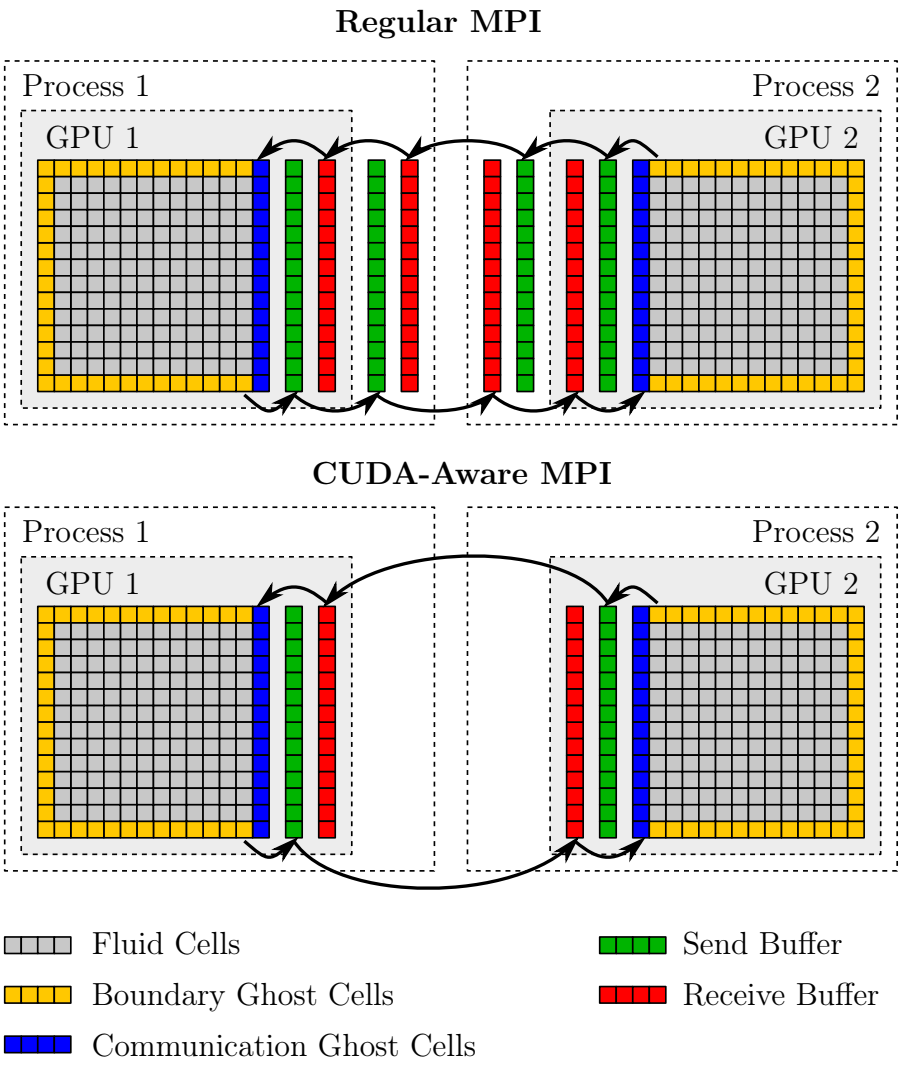


Figure 4.4: Multi-GPU communication pattern: Data is first collected on the GPU into send buffers. The send buffers are downloaded to the host and send via MPI, before they are uploaded to the device and distributed to the grid. CUDA-Aware MPI allows direct sending and receiving on the GPU, such that no buffers on the host are required.

for strong and weak scaling with N processes, respectively. The ideal speed up is usually $S = N$, where the communication has no influence on the performance. The parallel efficiency is the ratio of measured speed up over ideal speed up, i.e.

$$E = \frac{S}{N}. \tag{4.6}$$

The flow domain is split into as many sub-domains as GPUs are used. In the current version of the grid generator, these sub-domains can only be defined by cuboids. At the boundaries of the sub-domains data has to be exchanged between GPUs. The data exchange procedure is exemplary visualized in Fig. 4.4 for a simulation with

two GPUs. For the communication, the class `Communicator` is implemented, where each object is responsible for communication with one neighboring process. Each `Communicator` holds lists of indices of cells that should be send and that should be received. Further, each `Communicator` holds pointers to send and receive buffers on both host and device.

As a first step the data that should be send to the neighboring process must be collected in a single linear buffer, since the send cells are not continuous in memory. This is implemented by a kernel that copies the data from the grid into the buffer on the device. The list of send cells is generated by the grid generator, based on cell location. The send cells are the first plane of regular fluid cells inside the sub-domain. This buffer must then be downloaded to a similar buffer on the host, to allow MPI to send it to a receive buffer of the neighboring process. On the neighboring process the receive buffer is then uploaded to the device and scattered to the grid. The indices of the receive cells, i.e. the cells where the received data is scattered to, is also generated by the grid generator. The receive cells are the first plane of cells outside the sub-domain and the receive cells are ghost cells.

This procedure includes five copy operations per communication in one direction. Modern MPI implementations can be CUDA-Aware, which allows MPI to directly access GPU memory to send it to the neighboring processes, see Fig. 4.4. This saves memory and reduces the communication time.

The communication in more than one direction is performed sequentially. That means data is first communicated with neighbors in x-direction, then with neighbors in y-direction and finally in z-direction. This procedure allows information transport to diagonal neighbors, even though no direct communication with these neighbors takes place, see the highlighted cell in Fig. 4.5. The data is first send to the neighbor in x-direction, before it is send on in y-direction. The only requirement is that ghost cells are also send to the neighboring processes. In three dimensions this procedure works in the same way.

The final concept for Multi-GPU utilization implemented in VirtualFluidsGKS is communication hiding. In GPU computing the computation and the communication are performed by different hardware. Hence, it is possible to perform communication and computation concurrently. Two mechanisms enable the concurrent execution. First, CUDA-kernel calls are non-blocking, i.e. the host program continues execution immediately after kernel dispatch to the device. Second, device operations can be put into different streams to be executed concurrently. Memory operations (upload/download) can be done during kernel execution, because both involve different hardware that can work independently.

Nevertheless, communication hiding requires special care, because the computation is not logically independent of the communication. The received data in the communication ghost cells is required for the computation of the fluxes over the sub-domain boundary. Further, the faces normal to sub-domain boundary require the communication ghost cells for the reconstruction. This is why usually communication must be completed, before the computation is started. It is, however, possible to split the

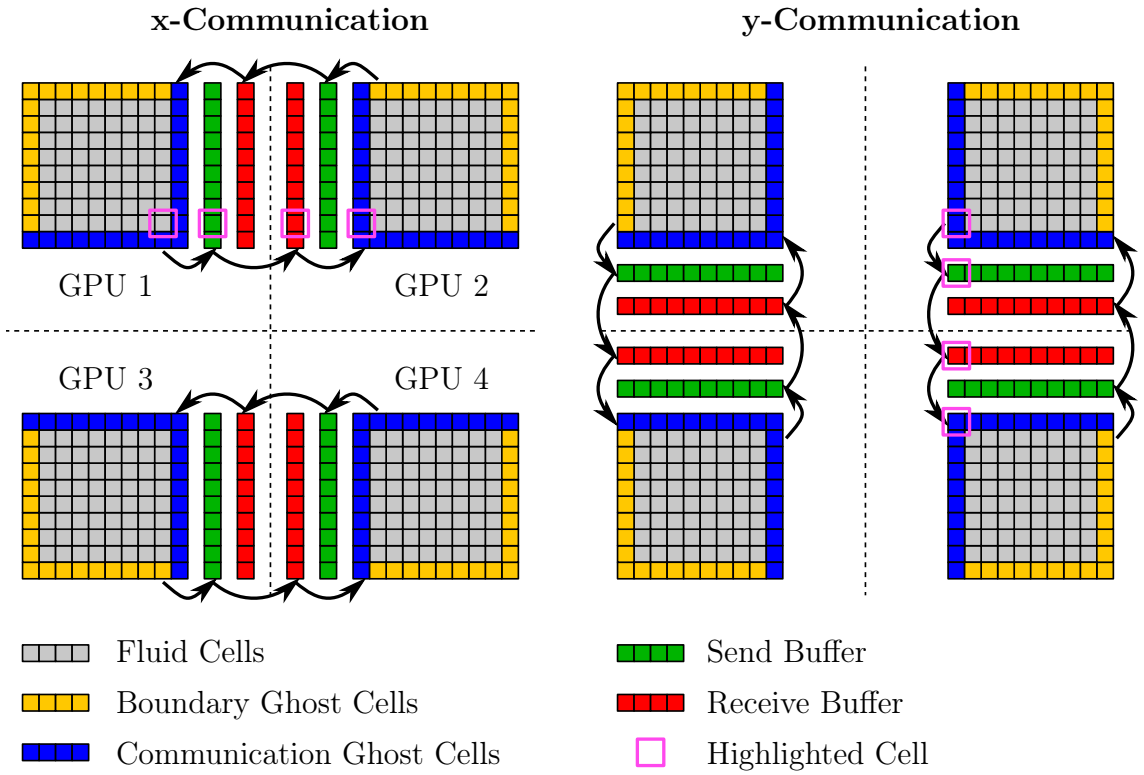


Figure 4.5: Multi-GPU communication pattern in two dimensions: The communication in the three directions is performed sequentially in the order x-y-z. The highlighted cell shows the information transport over diagonals.

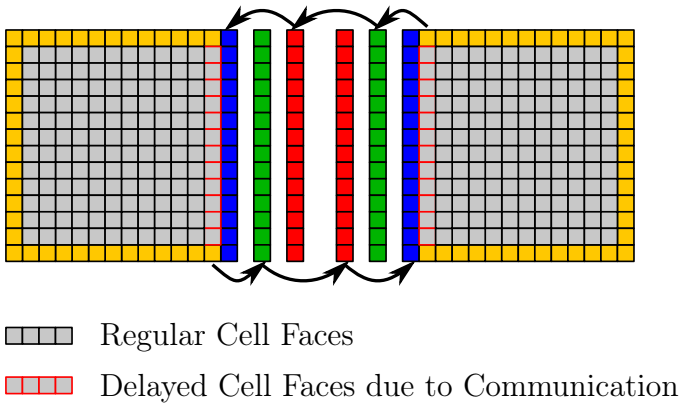


Figure 4.6: Multi-GPU: special cell faces for communication hiding

computation into two parts. For this reason, the cell faces have another numbering condition, such that regular faces, not dependent on the communication, come first and those which require data from the communication ghost cells come second. This distinction is shown in Fig. 4.6.

Algorithm 2 Recursive nested time stepping for Multi-GPU

```
1: function nestedTimeStep(level, Δt)
2:
3:     if level ≠ finestLevel then
4:         call fineToCoarseInterpolation(level), Eq. (2.167)
5:     end if
6:
7:     call setBoundaryConditions(level), see Section 2.5
8:
9:     call on Stream 0 computeFluxes(level, regularFaces), Eq. (2.102)
10:
11:    call on Stream 1 sendData(level, x)
12:    call on Stream 1 recvData(level, x)
13:
14:    call on Stream 1 sendData(level, y)
15:    call on Stream 1 recvData(level, y)
16:
17:    call on Stream 1 sendData(level, z)
18:    call on Stream 1 recvData(level, z)
19:
20:    call on Stream 1 computeFluxes(level, commFaces), Eq. (2.102)
21:
22:    synchronize device
23:
24:    if level ≠ finestLevel then
25:        call coarseToFineInterpolation(level), Eq. (2.169)
26:
27:        call nestedTimeStep(level + 1, Δt/2)
28:        call nestedTimeStep(level + 1, Δt/2)
29:    end if
30:
31:
32:    call updateCells(level), Eq. (2.136)
33: end function
```

The nested time stepping algorithm for Multi-GPU computations is shown in Algorithm 2. After fineToCoarseInterpolation and setBoundaryConditions the flux computation is dispatched to stream 0. All communication related tasks (collect and scatter kernels and memory copies) are dispatched to stream 1. First, data is send non-blocking to x-neighbors. Then the receive function is called, which blocks the execution, until the data is received. The same procedure is repeated for y- and z communication. In order to verify the concurrent execution, of the streams 0 and 1, the profiling tool nvprof, which ships with CUDA, is used. The output is shown in Fig. D.1 in the appendix.

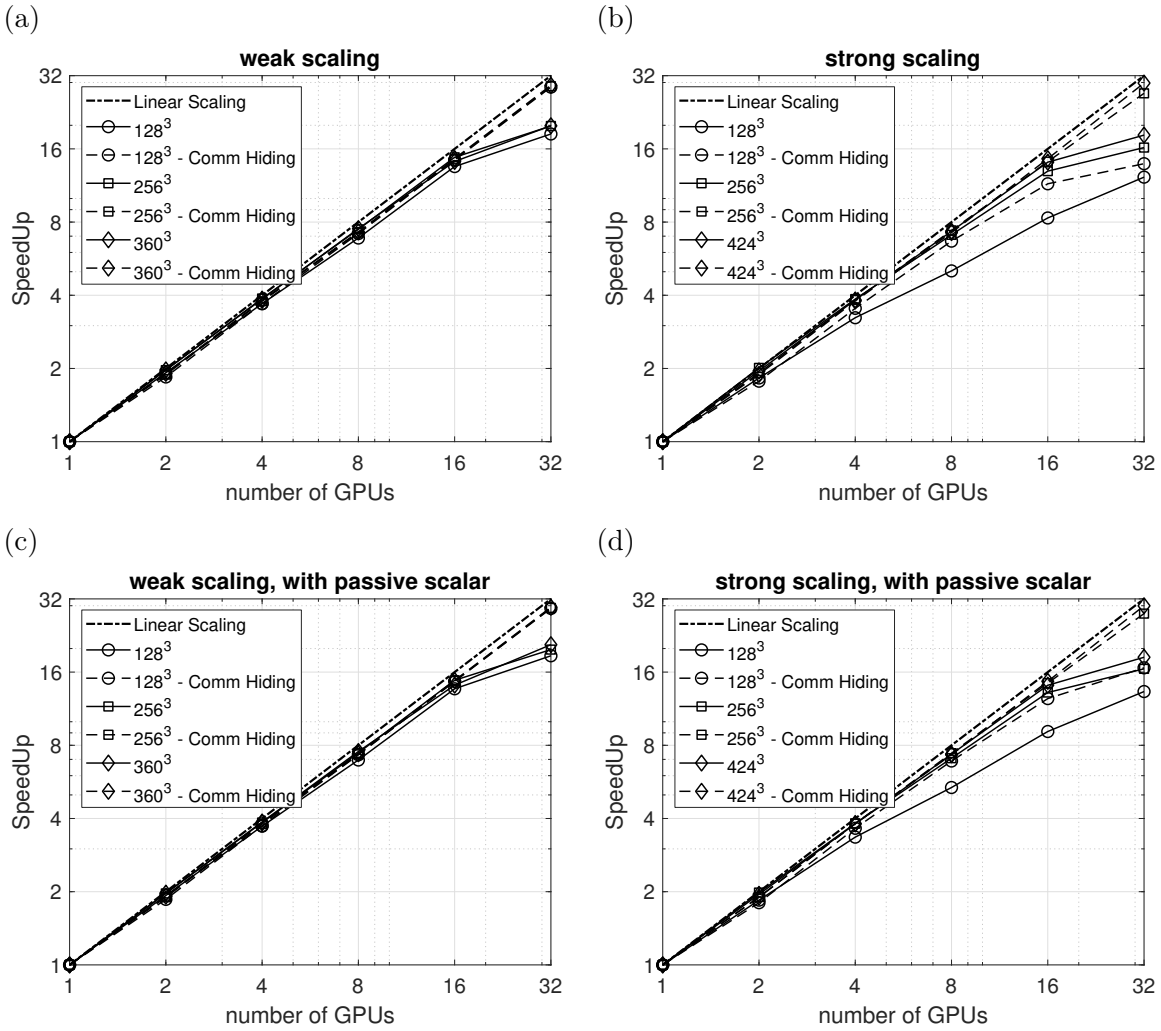


Figure 4.7: Multi-GPU: speedup

As a Multi-GPU performance test, a cubic domain with periodic boundaries is simulated. For strong scaling the domain is split into N parts, where N is the number of GPUs. For weak scaling the cube is repeated N times. The periodic boundaries are implemented by communication. Due to the cuboid type decomposition only powers of 2 are used for N. The decomposition with 2 GPUs is one-dimensional ($2 \times 1 \times 1$), with 4 GPUs two dimensional ($2 \times 2 \times 1$) and with 8 and more GPUs three dimensional ($2 \times 2 \times 2$, $4 \times 2 \times 2$ and $4 \times 4 \times 2$). The base domain is discretized by 128^3 and 256^3 cells. For strong scaling a third resolution of 424^3 cells is simulated. This resolution requires 95 % of the 16 GB of memory available on the used GPUs. Due to pre-processing restrictions, the maximal possible resolution for weak scaling is 360^3 cells.

The test was performed on the PHOENIX cluster at TU Braunschweig. Therein, eight GPU-nodes, with 4 NVIDIA Tesla P100 GPUs each, are available. The GPUs on the same node are connected with the fast NVIDIA NVLINK. Unfortunately, due to hardware/software incompatibilities the installation of CUDA-Aware MPI was not

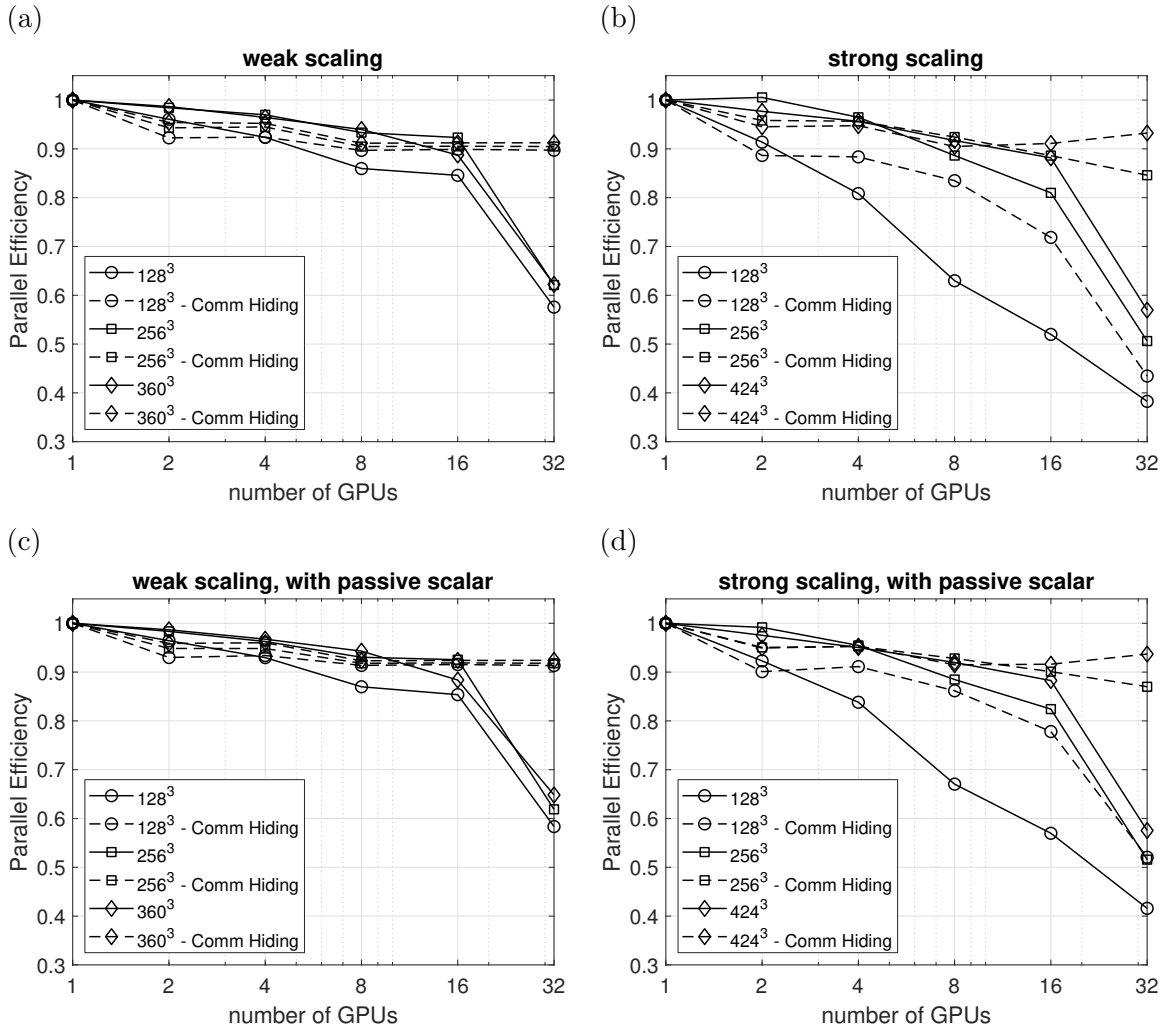


Figure 4.8: Multi-GPU: parallel efficiency

possible, such that the NVLINK could not be utilized. Hence, the PCI-E 3.0 bus was used for on-node communication. The interconnect between the nodes is an Intel Omnipath. The whole set of simulations was run through twice and the run times were added to reduce the impact of outliers, even though no outliers were observed.

The speedup S is shown in Fig. 4.7. All simulations show near to linear speedup up to 16 GPUs, while only the 128^3 simulation without communication hiding under strong scaling shows a performance loss for multiple GPUs. For 32 GPUs, the performance of all simulations without communication hiding drops drastically. Of the simulations with communication hiding this is only apparent for the 128^3 simulation under strong scaling.

The parallel efficiency E is shown Fig. 4.8. For weak scaling it is found that the communication hiding is not always beneficial. When using two GPUs the efficiency with communication hiding is below that without communication hiding. For the lower resolution and larger numbers of GPUs, the communication hiding increases

efficiency from about 85 % to above 90 %. The slight increase in efficiency from two to four GPUs is due to the amount of communication with 4 GPUs, which is twice as much than with only two. Hence, with four GPUs there is more communication to hide, while for two GPUs the overhead of splitting the flux computation prevails. From this argument one could expect a further increase in efficiency with eight GPUs, which is not found. This is because the eight GPUs are distributed over two nodes, where the slower communication over the interconnect comes into play. As already seen in the speedup, the efficiency drops drastically when going from 16 to 32 GPUs. Communication hiding is able to completely hide this effect.

For strong scaling a similar effect regarding communication hiding is visible at two GPUs, see Fig. 4.8 (b). At four GPUs it breaks even for all resolutions and at eight GPUs communication hiding is beneficial. Contrary to weak scaling, the overall parallel efficiency drops fast for more GPUs with out communication hiding and with a small number of cells. This is due to the increasing ratio of communication per computation for strong scaling. The efficiency gain by communication hiding is substantial. For the lower resolution communication hiding can lift the parallel efficiency by up to 20 percentage points. For the higher resolutions, communication hiding is able to keep the parallel efficiency at about 90 %.

The effect of the suddenly dropping efficiency from 16 to 32 GPUs can have several reasons. One aspect could be the hardware network topology in PHOENIX. The first five GPU-nodes are connected to one Omnipath switch, while the remaining three are connected to another one. The simulations with 16 GPUs were run on the first four GPU nodes, such that communication only had to pass one switch. When going to 32 GPUs, multiple concurrent communications have to be relayed through both switches, which could potentially have an impact on the performance. It was not possible, though, to reproduce this behavior with a smaller setup, which could be due to the smaller amount of required memory transfer.

5 Automated grid generation for GKS and LBM

Most LBM variants require uniform Cartesian grids to allow perfect shift. That means the discrete velocity couples grid spacing and time step. Further, the GKS in this work is implemented on uniform Cartesian grids. This chapter is dedicated to the generation of such grids.

Before the development of the present grid generator, grids for VirtualFluids-GPU were generate with LBMHexMesh [146], an extension of the OpenFOAM [147] grid generator SnappyHexMesh. Due to the dependence on OpenFOAM and difficult maintainability of LBMHexMesh, it was not developed further. Initial implementation work of a new grid generator was done by Sören Peters [148]. This present grid generator is based on the initial work of Peters and was developed in cooperation with the developer of VirtualFluidsGPU, Martin Schönherr [149].

5.1 Grid layout in LBM and GKS

In LBM the flow state is known at grid points. In GKS the flow state is known as cell average values, which are interpreted as cell centered values to second order of accuracy when needed. Hence, it is questionable how a grid generator can produce grids for both methods. A uniform Cartesian grid can be interpreted in terms of cells in two ways. First, the grid points are the cell vertices. The information location between grid points and cell centers is shifted by half a grid spacing. Alternatively, the grid points can be interpreted as cell centers, such that the information location coincides with the grid points. The second variant is chosen here. At the boundary this second approach is also consistent. The simple LBM bounce back boundary condition is only second order accurate, if the wall is half a grid spacing from the first grid point [28]. In GKS the cell around the first grid point extends for the same half grid spacing towards the wall, such that the cell face is located exactly on the boundary. Hence, the grid generator presented here operates on grid points, that are finally interpreted as cells in GKS or lattice nodes in LBM.

In LBM complex boundaries, i.e. boundaries that are not parallel to the grid axes and boundaries that have not half a grid spacing distance to the first grid point, can be computed based on sub-grid distances. LBM operates on a discrete particle velocity space, where each discrete velocity is one lattice link, i.e. it connects the current node with one neighbor node. If a lattice link intersects the boundary, its sub-grid distance

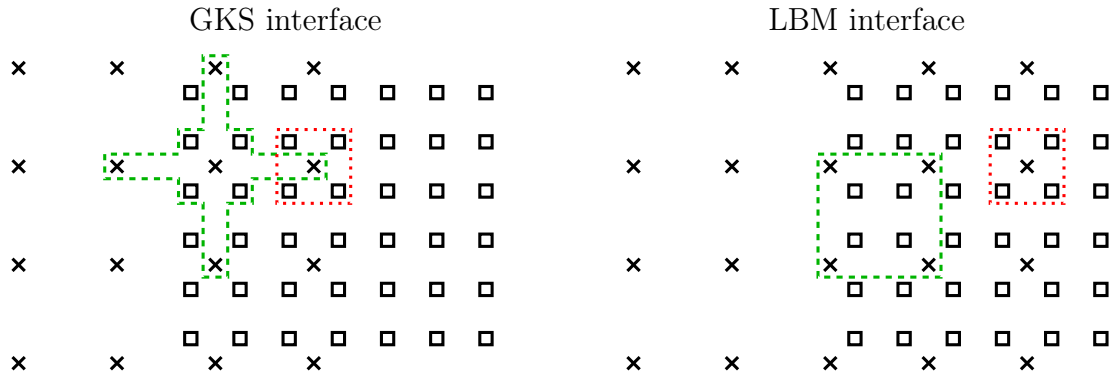


Figure 5.1: Interpolations stencils of GKS and LBM. The coarse to fine stencils are green and the fine to coarse stencils are red.

is the part of the lattice link from the grid point to the boundary. This approach is similar to a cut-cell approach where the intersection is only performed along a low number of lattice links resulting simple shaped cut-cells. Based on this a morph-cell algorithm for GKS is proposed in Section 5.3.

For both LBM and GKS algorithms for coupling octree based grid levels exist. In GKS the arrangement of cells is trivial. A coarse cell is split in eight fine cells, see Fig. 2.7 for a two-dimensional analog. Hence, the grid points on coarse and fine grids do not coincide, but are shifted by half a fine grid spacing. Choosing the same grid for LBM results in a staggered grids. In LBM grid refinement is often done on grids with coinciding grid points [109, 150]. The refinement proposed by Geier et al. [110], which is implemented in VirtualFluidsGPU uses a staggered grid. Hence, the relative location of coarse and fine grids is identical for GKS and LBM.

The interface interpolation stencil is not similar, though. The explicit knowledge of non-equilibrium information in LBM allows the computation of gradients from the distribution function. By this, it is possible to construct a second-order accurate interpolation on eight nodes, that form a cube. In GKS this is not possible, such that three points per coordinate direction are required. The resulting stencil is wider, but requires only seven points in total. The two-dimensional interpolation stencils are shown in Fig. 5.1.

5.2 Grid generation algorithm

In this section the grid generation algorithm is introduced. The algorithm is explained based on a simple two-dimensional example in Figs. 5.2 and 5.3. In this example a square domain with solid cylinder is considered. The grid around the cylinder is refined once.

5.2.1 Data structures

The grid generator is based on a full structured grid. The main data of the grid is a classification of the grid points by types. Each grid point has a type and all the following algorithms operate on these types. Each grid level has its own field of grid point classifications. The types are:

- *fluid* for regular fluid grid points
- *fluid_cfc*, *fluid_cff*, *fluid_fcc* and *fluid_fcf* for interface points, where the first two characters denote coarse to fine and fine to coarse interpolation stencils, and the third character, whether the point itself is on the coarse or fine grid
- *bc_pressure*, *bc_velocity* and *bc_solid* for different boundary conditions
- *stopper_out_of_grid*, *stopper_coarse_under_fine*, *stopper_solid*, *stopper_out_of_grid_boundary* for stopper nodes
- *invalid_out_of_grid*, *invalid_coarse_under_fine*, *invalid_solid*

The stopper types denote the first layer of points outside the domain. For GKS these are candidates for boundary condition ghost cells, see Section 2.5. For LBM these points are required for the Eso-Twist streaming pattern [151], where information is stored on neighbor nodes. The invalid grid points are discarded after grid generation.

5.2.2 Grid initialization

The first step is the initialization of the grids on all levels, see Fig. 5.2 (a). The domain on the coarsest level is defined by a cuboid bounding box and a grid spacing. The base coordinates of the grid are shifted by half a grid spacing to the outside to allow for one layer of stopper nodes.

The grid for the fine level is defined based on the cylinder and a number of layers (8 here) of cells around the cylinder, where the cylinder is defined by a triangulated surface. The number of layers is related to the minimum number of fluid grid points. Two additional layers are added to allow for complete interpolation stencils and stopper nodes. In the fluid domain, the fine grid exceeds the last overlaid coarse grid point by a single fine point. At the boundary of the domain the first fine grid point is inside the outer most coarse grid point (not shown in the figure). All grid points are initially set to *invalid_out_of_grid*.

In addition to the refinement shown in this example, the refinement region can be defined by primitives, such as cuboids or spheres. The generalization of primitive shapes and triangulated surfaces is called object in this grid generator. The object has to define methods to extract a bounding box, i.e. maximal and minimal coordinates in all directions as used for the size of the grid. Further, the object has to define a method for an inside-outside check, i.e. that returns true, if a query point is in the

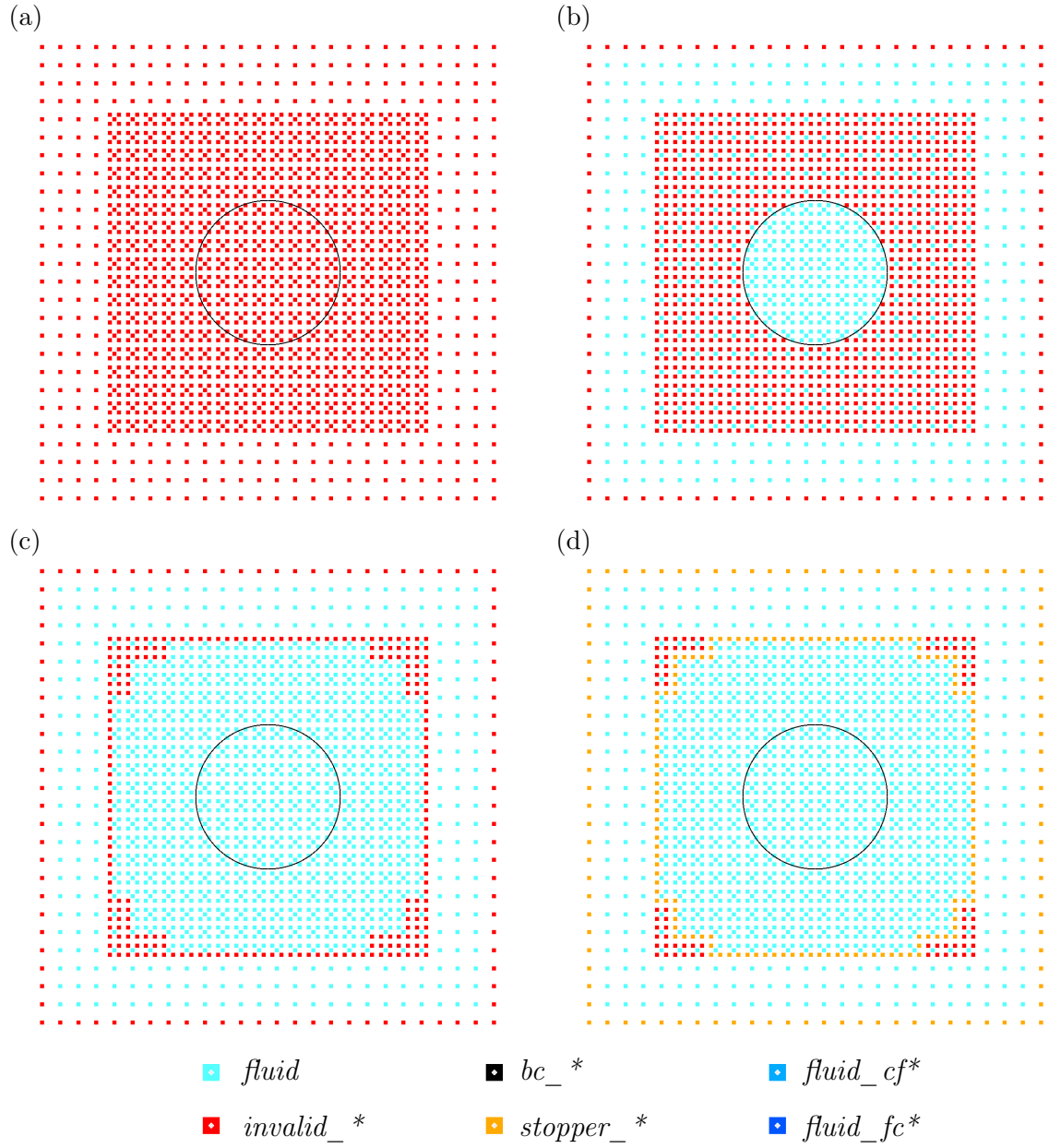


Figure 5.2: Grid generator example: grid initialization

object. For primitive objects this is usually trivial. For triangulated surfaces this is done by a ray-tracing based point-in-object test. This ray-tracing code is common in all VirtualFluids codes and was developed as part of VirtualFluidsCPU [152, 153]. Further, several objects can be combined by a conglomerate object.

Next, fluid points are identified. On the coarse grid no inside-outside check is required, because every thing except the outer most layer is inside. It is, hence, set to *fluid*, see Fig. 5.2 (b). On the fine level, the *fluid* points are identified by the inside-outside check of the object. For multiple levels the inside region can also be identified by the next finer level.

Then layers are added to extend the refinement region, see Fig. 5.2 (c). This is implemented by a double sweep over the domain, where each grid point that has a *fluid* neighbor is first set to a temporary type and in the second sweep set to fluid. By being layer based, this procedure results in cuboid regions for many layers and is, hence, different than distance based methods. This algorithm can be extended to distance refinement by additionally storing Cartesian distances. This extension was tested in two and three dimensions, but not added to the grid generator, due to substantially increased memory demand.

As the last step of the initialization, the stopper nodes are identified, see Fig. 5.2 (d). On the coarse grid, the stoppers are set to *stopper_out_of_grid_boundary* and on the finer grids to *stopper_out_of_grid*.

5.2.3 Solid domains and sub-grid distances

The next stage in the grid generator concerns the definition of solid domains. In principle solid domains can be defined by objects, as introduced in the prior section. In practice, this currently only holds for GKS in three dimensions, where no sub-grid distances are required at the time of this writing. For LBM and for GKS with morph-cell boundaries, the solid domain must be defined as a triangulated surface. The computation of sub-grid distances is only implemented for triangular surfaces, because of a corresponding demand. The implementation of the sub-grid distance computation for primitives was not yet required, but should in principle be straight forward.

For GKS grids in three dimensions, the solid object can extend over multiple grid levels. For LBM grids, the solid object must be completely on the finest level, at the time of this writing.

As a first stage in the consideration of solid domains, an inside-outside check for the solid object is performed, see Fig. 5.3 (a). This inside-outside check is the same as performed for the definition of the refinement region, but sets the inner grid points to *invalid_solid*. After this, fluid cells that have two solid neighbors in at least one coordinate direction are closed, i.e. set also to *invalid_solid*. Such a "close needle cells" fix was also implemented in LBMHexMesh [146]. Subsequently, stopper nodes in the solid region are set to *stopper_solid*. Finally, the fluid nodes around the solid region are identified by having solid stopper neighbors and are set to *bc_solid*. LBM boundary conditions will be applied on these nodes.

Thin walls, i.e. walls that are thinner than $\sqrt{3}\Delta x$, are not found correctly by this approach, because it is possible that the solid domain is in between two adjacent grid points. These two grid points would not be found as inside the solid object. Hence, if thin walls should be considered in the grid, a second approach to identify *bc_solid* grid points is implemented. This approach is based on computation of sub-grid distances.

The sub-grid distances are computed by the same ray tracing algorithm used for the inside-outside check of triangulated surfaces [152]. The algorithm iterates over all

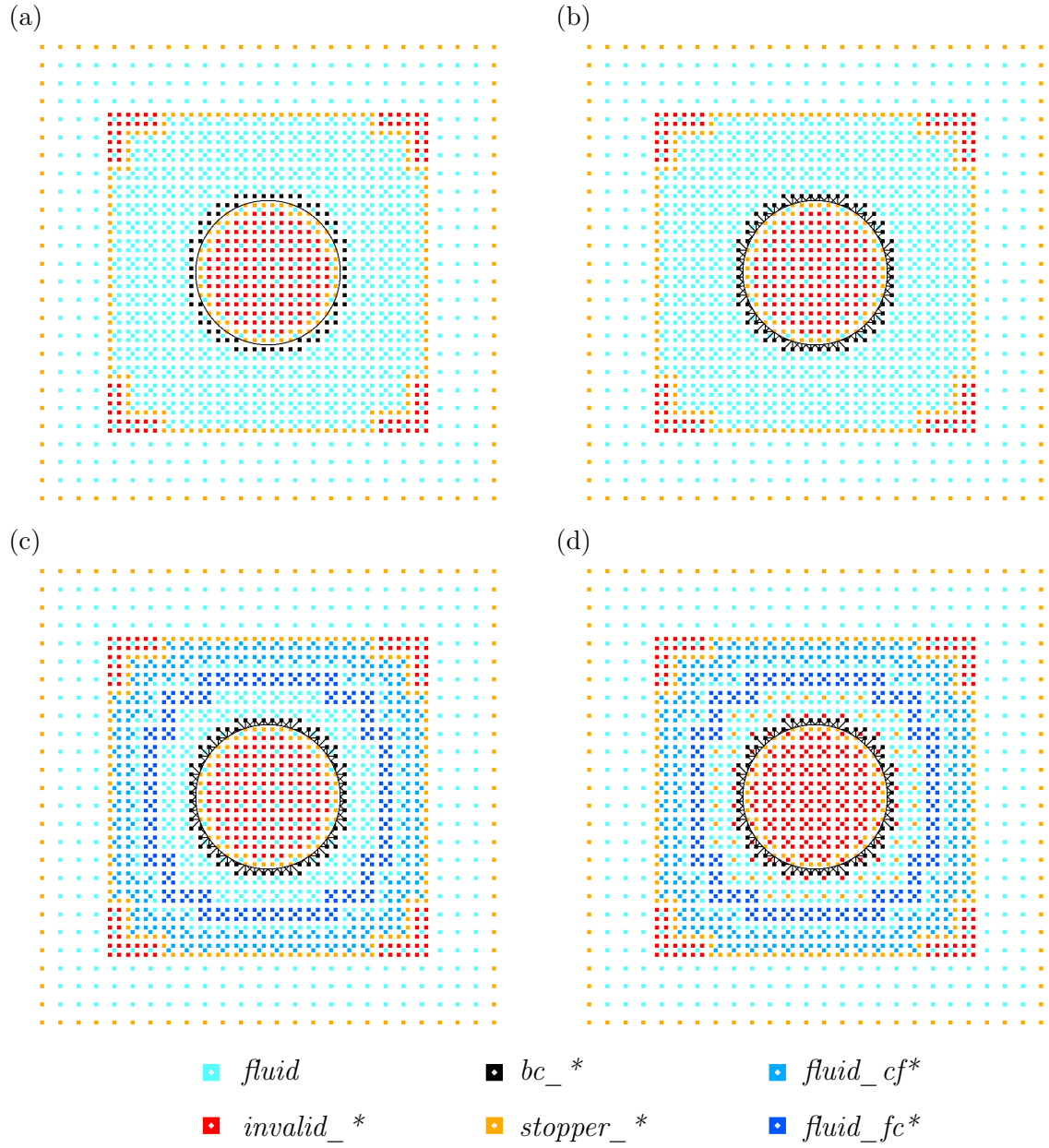


Figure 5.3: Grid generator example: solid objects and interfaces

triangles of the triangulated surface. It, therein, iterates over all grid points in a bounding box of the triangle. Finally, it iterates therein over all lattice links and computes the intersection of the lattice link with the triangle. If the lattice link intersects the triangle, the current grid point is also set to *bc_solid*. It is worthwhile to note that such grid points are not guaranteed to have stopper neighbors.

At this point, the number of *bc_solid* grid points is counted and memory for their sub-grid distances is allocated. This includes an index map to map between the sub-grid distance fields and the full grid. For each *bc_solid* grid point 27 sub-grid distances have to computed and stored, such that allocating memory for the whole

grid would waste much memory. Then the computation of the sub-grid distances is repeated. This time only *bc_solid* grid points in the bounding box of the triangles are considered, to reduce the work load. Further, this time the sub-grid distances are stored. This second run is independent of the first one. Hence, the first run is only activated if thin walls should be considered. The computed sub-grid distances are shown in Fig. 5.3 (b).

5.2.4 Interface interpolation stencils

At this point, the grids were mostly processed independently. In this section the construction of the interpolation stencils at the grid interfaces is discussed.

First, the interface from coarse to fine is constructed. The stencils for LBM and GKS are shown in Fig. 5.1. The coarse to fine stencil for LBM can be addressed by the indices of the points on the bottom left (i.e. the lower point in all three coordinate directions) on both coarse and fine grids. The remaining seven points on each level are addressed as direct and indirect neighbors of these two points. These grid points are identified by a sweep through the coarse grid. The grid point on the coarse grid must be classified as *fluid* or *stopper_out_of_grid_boundary*. It must have a valid corresponding grid point on the fine grid, i.e. a grid point on the fine grid with a quarter of a grid spacing in all coordinate directions. This point on the fine grid must also be *fluid* or *stopper_out_of_grid_boundary*. If these conditions are met, the algorithm checks, if the coarse neighbors exist and are not stoppers. Then it checks whether these neighbors have corresponding fine points that either do not exist, are *invalid_out_of_grid* or *stopper_out_of_grid*. In this case this coarse node defines an interpolation stencil. All eight points on the coarse grid and the eight points on the fine grid are set to *fluid_cfc* and *fluid_cff*, respectively, if they were *fluid* before. In the vicinity of the domain boundaries, this stencil might contain stopper or invalid grid points and, hence, be incomplete. In that case an offset of the interpolation stencil is used, such that the fine grid points are extrapolated from a complete stencil in the domain [149]. These offsets are found by first checking whether one of the face neighbor points on the coarse grid defines a full stencil, then of the edge neighbors and finally of the corner neighbors.

Due to the different layouts of LBM and GKS coarse to fine interpolation stencils, the construction is also different. For GKS the coarse grid point is only required to be *fluid*. The algorithm then checks, whether a neighbor on the fine grid is *stopper_out_of_grid*. In this case the coarse grid point is set to *fluid_cfc*. For GKS the indices are not stored and the fine grid point types are not modified. This information is collected in the adapter that transfers the grid from the grid generator to the GKS data structures, see Section 4.2.5.

The fine to coarse interpolation stencil in LBM and GKS is similar. The corresponding grid points are found based on the earlier constructed coarse to fine classification. The grid point of the coarse grid must be *fluid*. The coarse grid point must also have a corresponding fluid point on the fine grid, this time looking in negative direction.

Finally the coarse grid point is part of the fine to coarse interpolation stencil (see Fig. 5.1), if it has neighbors that are classified as *fluid_cfc*. The indices are stored and the grid points are set to *fluid_fcc* and *fluid_fcf*. The interpolation stencils for LBM are shown in Fig. 5.3 (c) and for GKS in Fig. 5.4 (b).

At this point, all interface interpolation stencils are constructed. Next, the stopper nodes on the coarse grid are found similar as the fine to coarse interpolation stencils, this time checking whether the coarse neighbor is *fluid_fcc*. In the same sweep, the coarse grid points that are overlapped by valid fine grid points can be set to *invalid_coarse_under_fine*, see Fig. 5.3 (d). Offsets are not implemented for the fine to coarse interpolation, because currently the grid generator does not support refinement into a solid domain boundary. For GKS refinement into the solid domain is allowed, because offsets are not required. This is because the stopper nodes are interpreted as ghost cells that carry flow state information, which can be used in the interpolation. If the refinement extends into the domain boundary, more stopper nodes can be invalidated for GKS.

At this point the classification of cells is complete.

5.2.5 Grid finalization

The last step in the grid generation algorithm is discarding the invalid grid points. For GKS is is done in the adapter, see Section 4.2.5. For LBM a field of sparse indices of the full size is allocated. Then in a single sweep the sparse indices are counted through the grid, where all invalid grid points are skipped. Further, neighbors are searched in terms of sparse indices. The neighbors are required for indirect addressing and the unstructured grids used in the simulations. In this process the grid generator handles periodic boundaries by setting neighbors to the opposing side of the domain, if activated by the user. Further, the indices in the interface interpolation stencils are updated to sparse indices.

The last step in the grid generations is the definition of boundary conditions. For GKS this is done as described in Section 4.2.4 and based on the adapter. For LBM the boundary conditions are defined in the grid generator. Different boundary conditions can currently be applied to the six faces of the cuboid of the background grid and to the solid, where the latter grid points are already classified. Depending on the type of boundary condition, the grid points on the boundary are set to *bc_velocity* or *bc_pressure*. Internally, the grid generator holds three separate lists with indices of boundary grid points. Further, it is possible to set different velocity and pressure values for each of the six sides.

The final grids for LBM and GKS are shown in Fig. 5.4 (a) and (b) for LBM and GKS, respectively. The following three subsections introduce three special procedures that are also implemented in the grid generator.

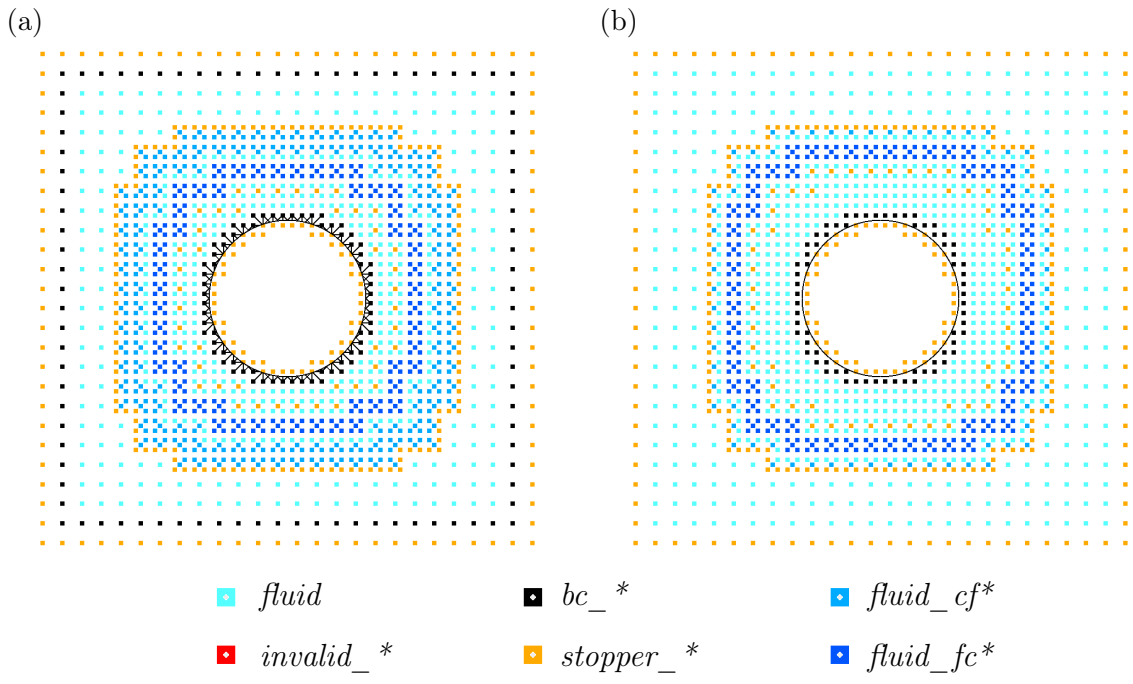


Figure 5.4: Grid generator example: final grids for LBM (a) and GKS (b)

5.2.6 Fix refinement into the wall

If the refinement extends into the domain boundaries it is important to do so in normal direction, such that fine and coarse levels, both have a minimum normal extend to the boundary. To check this, the algorithm iterates over all nodes on the boundary and ensures that the normal extend is satisfied. This procedure is performed directly after the layers are added, see Fig. 5.2 (c).

5.2.7 Rotating velocity boundary condition

For engineering CFD it happens that the solid domain is not stationary. In the external aerodynamics of cars, for instance, the wheels rotate. In such cases the velocity on these boundary grid points is not constant but depends on the location of the grid points. The solid geometry is usually defined as a triangulated surface stored in an STL (Standard Triangulation Language) file. Therein, the definition of groups of triangles is possible. When read into the grid generator, it is first possible to ignore groups. Further, the grid generator stores the group index per triangle. When the *bc_solid* grid points are constructed, these groups are also stored for these grid points. Hence, the grid points that belong to a specific group can be identified. After the classification, the velocities on the rotating object are computed from a rotation axis and a rotation frequency. This addon is only implemented for LBM.

5.2.8 Domain decomposition

The topic of domain decomposition was introduced in Section 4.4 for Multi-GPU computations. Here, the grid generation aspect of the domain decomposition is discussed. The grid generation is distributed in the same way as the computation is. Each instance of the grid generator (usually one instance per MPI process) generates one part of the grid. To this end, a bounding box of the sub-domain is defined. The coarse grid extends this sub-domain box by a minimum of three coarse grid spacings. The grid is then generated exactly as described above. After the interface interpolation stencils are constructed, the domain is limited to the sub-domain. In this process first all grid points that are not in the sub-domain are set to *stopper_out_of_grid*. Then for GKS all but one layer of grid points on the boundary of the sub-domain are invalidated to *invalid_out_of_grid*. For LBM two layers are left over, because in LBM receive nodes plus stopper nodes are required due to the Eso-Twist streaming pattern [151]. In this process also lists of send and receive grid points are generated.

5.2.9 Examples

In the last sections the functionality of the grid generator was introduced. In this section, three examples for these functionalities are given. The simulations in this section are performed with VirtualFluidsGPU, because VirtualFluidsGKS does not support the functionality shown in this section. Examples for grids generated with the grid generator for VirtualFluidsGKS, are found over the course of this work, see for instance Figs. 6.24 and 6.30.

The first example demonstrates the flow around two finite cylinders. The blue cylinder is resting, while the red cylinder is rotating, see Fig. 5.5. The boundary conditions are velocity on the upwind wall and all side walls and outflow on the downwind wall, such that the cylinders are subject to free flow. The free flow Reynolds number based on the cylinder diameter is $Re = 100$. On the resting blue cylinder, the stream lines pass the cylinder in a symmetric fashion. On the rotating red cylinder, the fluid is dragged below the cylinder following the rotation. Behind the cylinder the fluid is further dragged up. This has an implication on the lift of the cylinder. While the blue cylinder has neutral lift, the red cylinder will have negative lift, due to the Magnus effect [154].

The second example demonstrates the thin wall boundary condition. The set up is again free flow with a solid obstacle. In this example the obstacle is a paper plane, made up of four planar sheets of paper, see Fig. 5.6. The length of the paper plane is 0.3 m and the paper thickness 10^{-4} m. The length based Reynolds number is $Re = 90,000$. The grid has a total of four levels with a resolution on the paper plane of $\Delta x = 0.00125$ m. Hence, the thickness of the paper is more than an order of magnitude thinner than the grid spacing. Due to the absence of invalid grid points between the boundary nodes, the triangulated surface of the paper plane cannot be used to define the refinement regions. Hence, a second triangulated surface for refinement has to be constructed.

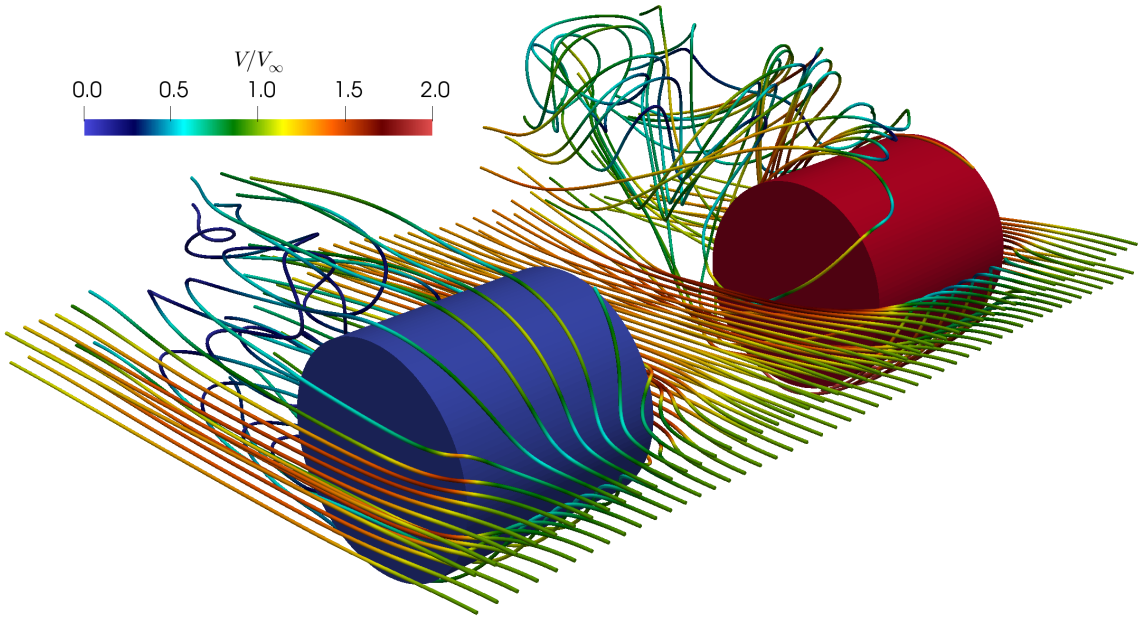


Figure 5.5: Flow around two cylinders at $Re = 100$: The blue cylinders is resting and the red cylinder rotates.

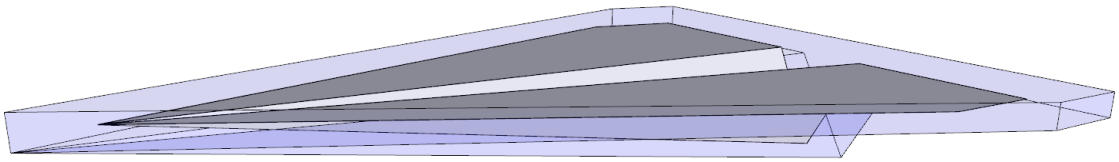


Figure 5.6: Geometry of the paper plane including the triangulated surface for the refinement

A slice of the generated grid is shown in Fig. 5.7. It is found, that the black grid points (*bc_solid*) on top and bottom of the wings are directly adjacent. The sub-grid distances (black lines) go from points on both top and bottom towards the wing. In order to show that the walls are actually not penetrated, streamline of the simulation are observed, see Fig. 5.7.

The last example considers the external aerodynamics of a the generic DrivAer car model [155]. The car is described by a triangulated surface with more than $600,000$ triangles. The grid is constructed with a coarse background mesh with $\Delta x = 0.2\,\mathrm{m}$. A triangulated surface that follows roughly the geometry of the car, but is extended in the wake of the car, is used to define four refinement levels. The fifth refinement level is defined by the geometry of the car itself. The resolution on the surface of the car is $\Delta x = 0.00625\,\mathrm{m}$. The resulting grid is shown in Fig. 5.8. It is evident, how the grid follows the shape of the car. The wheels have a rotation velocity. The simulation features a Reynolds number of $Re = 1,000,000$ based on the length of the car. The

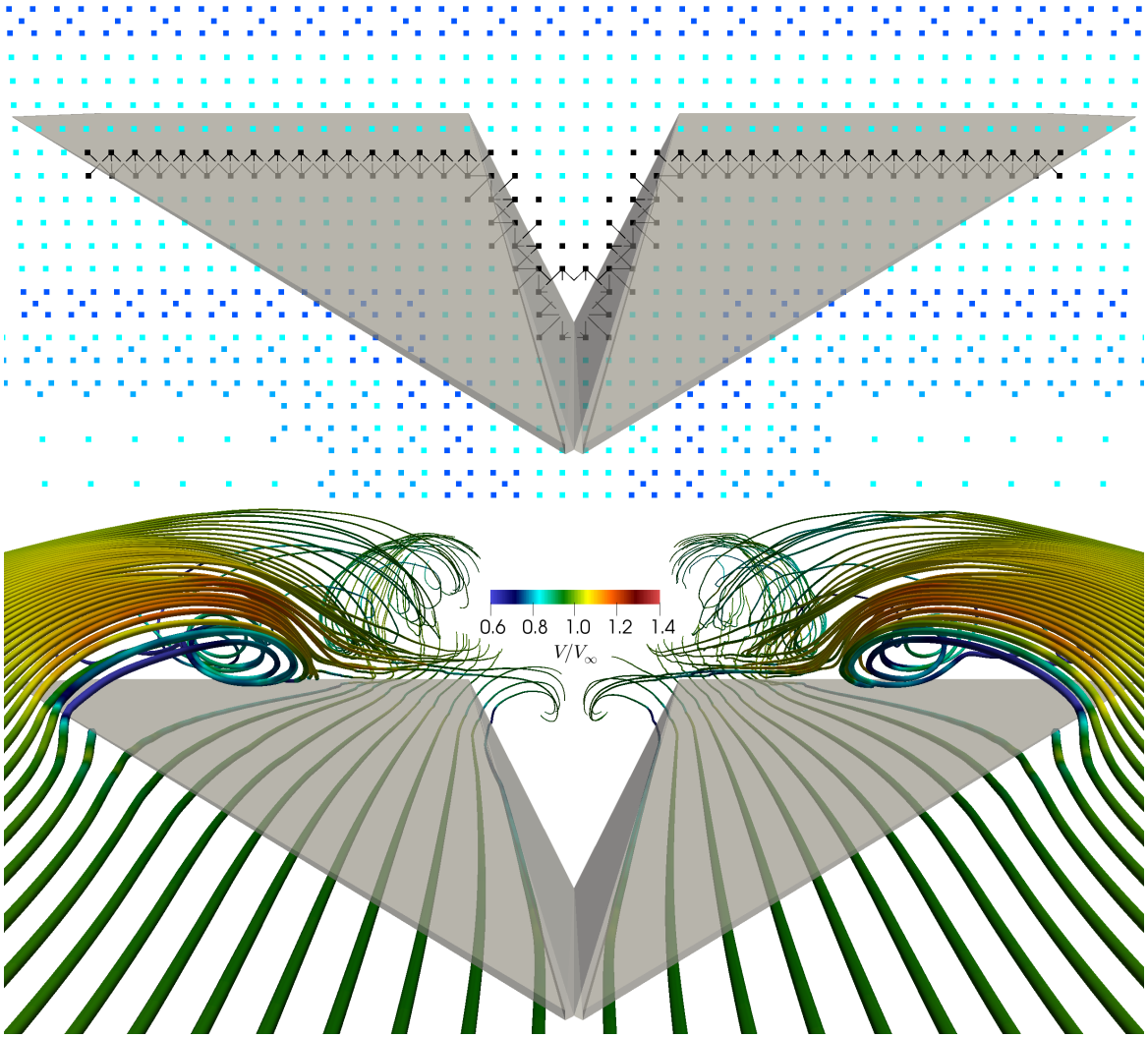


Figure 5.7: Grid and streamlines of the paper plane simulation

normalized velocity field at the mid plane is also shown in Fig. 5.8.

5.2.10 Limitations

The current implementation of the grid generator has several limitations, which will be discusses in this section.

A first limitation is rooted in the data structures. The reliance on a structured full matrix bounding box grid for the classification is problematic, if the fine levels have few grid points spread over the whole domain. In this case the fine grid has to extend over the whole domain, even though only a very small number of these grid points will be valid. Examples for this limitation are the cavity flows, see Sections 6.2.2 and 6.3.1. Therein, the problem is bypassed, by using Multi-GPU for both simulation and grid generation, such that only one of the two walls is present per instance of the grid generator. In this case, the fine grid only has to exist close to walls. A remedy

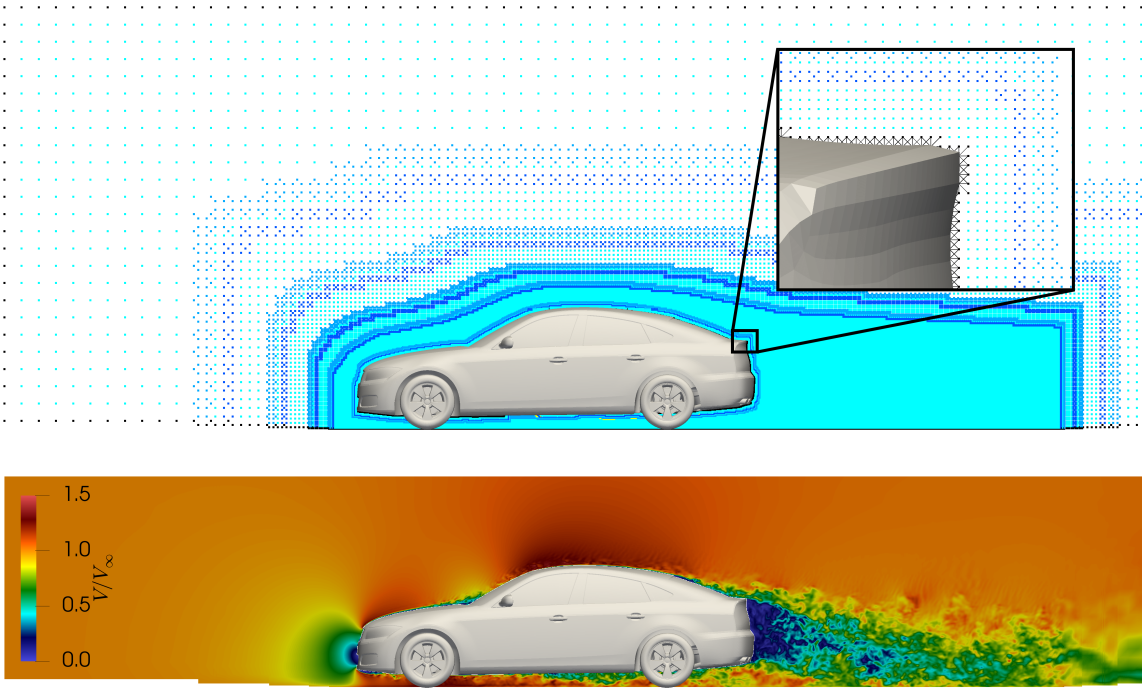


Figure 5.8: Grid and velocity field for the external aerodynamics simulation of the DrivAer model

for this inefficiency is not in sight, because the algorithms utilize the structured grid, extensively.

Further, limitations of the grid generator are related to the definition of solid domains in the flow domain. First, the computation of sub-grid distances on primitives is not implemented. Second, it is currently not possible to have a solid domain on another then the finest level. In other words, the coarse to fine interface cannot intersect the solid boundary. In the current version, this is only implemented for the domain boundaries. Furthermore, periodic boundaries can currently only be on the coarsest level. These features are mainly missing, because they were not required up to this point of the development. Contrary to the first limitation in this section, the grid generator should allow for these extensions.

Finally, Multi-GPU is implemented in a straight forward way, see Section 5.2.8. We are currently aware of two limitations caused by this approach. When the shape of the refinement region is very complex, we observed that the grid generator does not behave similar in both domains, such that the number of communication indices does not match. This problem can probably be solved in the current framework. A bigger limitation is the scalability of this type of domain decomposition. Currently, all grid levels are decomposed together, based on a geometric split. The distribution of the grid points on finer levels will not be uniform over the domain, though. The fine levels will usually be clustered around the area of interest, while the coarse levels extend to the far field. Using the present decomposition with many domains will lead to a very poor load balancing. This effect is amplified by the nested time stepping. In

order to improve the scalability, the levels have to be distributed independently with additional communication for the grid interfaces. Further, this distribution must take the complex topology into account. Both points are implemented in the block based VirtualFluidsCPU, where the metis library [156] is used to distribute blocks of points. At this point it is not clear how better scalability for the non-block based GPU versions of VirtualFluids can be implemented.

5.3 A novel morph-cell algorithm

For LBM the complex solid boundaries are incorporated into the solver by the means of sub-grid distances. VirtualFluidsGKS cannot take these into account. Inclined boundaries will, hence, have a staircase geometry. In this section a novel morph-cell algorithm is discussed, which allows to enable complex boundaries in GKS. This algorithm is only developed in two dimensions, such that it is not part of VirtualFluidsGKS at the time of this writing.

In the finite volume method boundary conforming grids are usually generated in the frameworks of arbitrary unstructured grids. For Cartesian grid methods cut-cell approaches exist, e.g. [157, 158, 159, 160]. A drawback of cut-cells is that the final shape of the cells may be complicated. The morph-cell approach proposed here is a compromise between cell simplicity and accuracy of boundary representation. The basic idea is to utilize the sub-grid distances to morph the cells, such that they conform with the boundary.

5.3.1 Generation of boundary conforming morph-cells

The morph-cell algorithm is visualized in Fig. 5.9. We start with the grid generated by the grid generator, where the grid points are the intersections of the blue grid lines, see Fig. 5.9 (a). The sub-grid distances are shown in green and red. For the morph-cell algorithm only the sub-grid distances that point through a cell vertex are used (green). First the grid is overlaid with the cell centered finite volume cells, see Fig. 5.9 (b). Then the grid is discarded, see Fig. 5.9 (c).

The actual morph-cell step is shown in Fig. 5.9 (d). For each boundary cell the vertices that have a corresponding sub-grid distance are moved to the tip of the sub-grid distance. By this, the vertices are guaranteed to be on the surface. When two cells want to move the same vertex the final location is averaged. If the two sub-grid distances point to the same triangle this still preserves the surface. If they are on different triangles, the final point is not exactly on the boundary any more. The resulting cells of this operation are still guaranteed to be quadrilaterals. Some cells will be very close to triangles or even be concave. These cells are repaired to triangular cells, see Fig. 5.9 (e).

New ghost cells are constructed for each cell face of the morphed boundary cells, that do not have a valid neighbor, see Fig. 5.9 (f). Their centers are mirrored on the cell

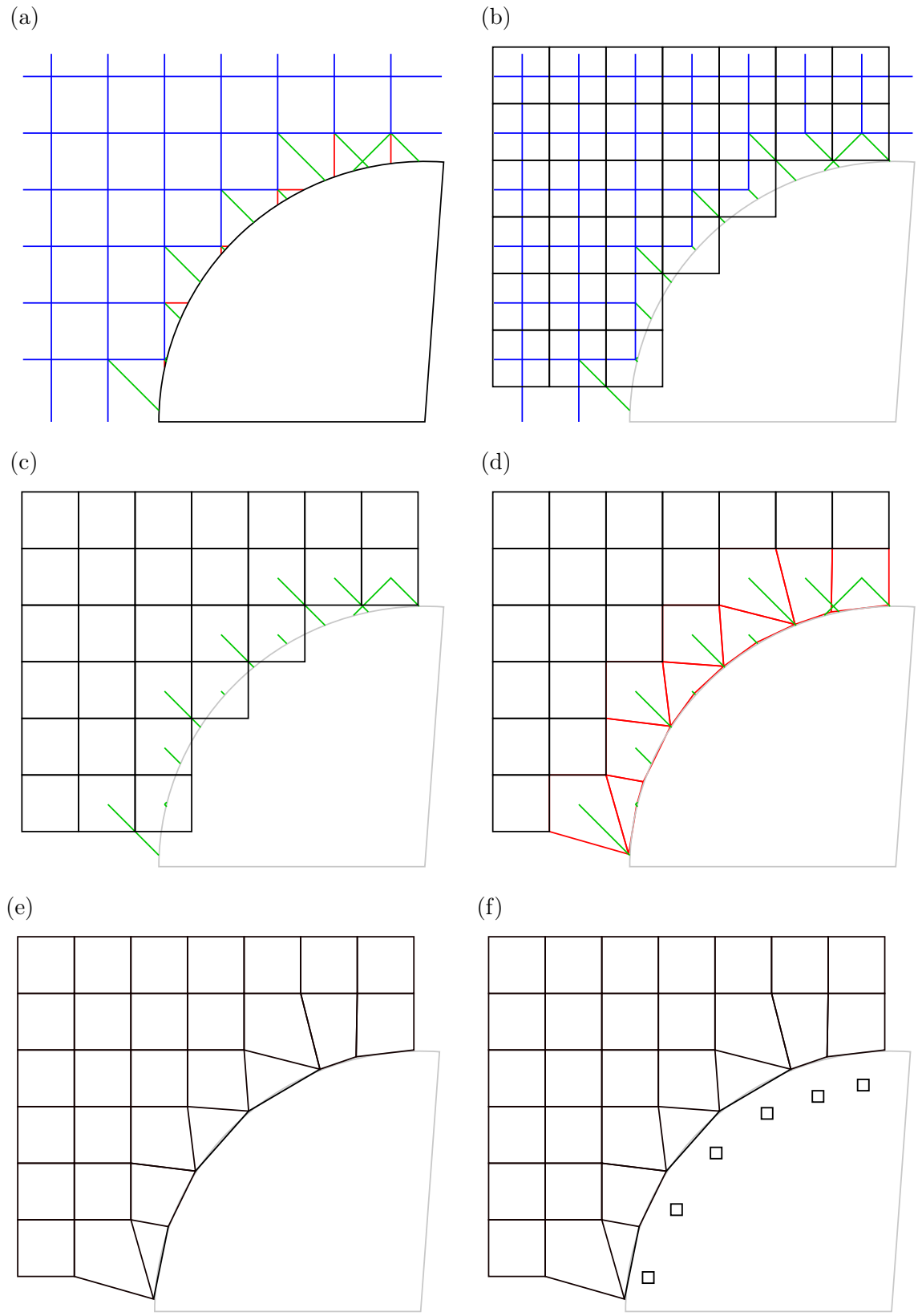


Figure 5.9: Generation of boundary conforming morph-cells: The cell vertices are moved to the tips of the sub-grid distances.

face centers, such that the three centers of boundary cell, ghost cell and cell face are aligned. Finally, the geometric values of the boundary cells are computed based on [101, Chapter 5]. This includes cell center locations, cell volumes, cell face areas, cell face centers and cell face normals.

A common problem in cut-cell approaches is that the size of cells can be arbitrarily small, such that the CFL criterion for stability cannot be satisfied [159]. Proposed remedies include merge-cell approaches [161], where the small cell is merged with neighboring full size cells, and master-slave approaches, where the small cells are slaved to full size master cells [162]. The present morph-cell algorithm avoids this problem by discarding all cells with centers in the solid domain. Hence, half the cell is in the flow domain. The cell morphing will not reduce the cell size indefinitely such that there is not stability issue with small cells in this approach.

5.3.2 Flux computation on morph-cells

The grid around the boundary after morphing the cells is not Cartesian or uniform anymore. Hence, the reconstruction introduced in Section 2.4.3 is not valid.

Two options are considered for the computation of the face state $\underline{W}$. First, the face state can be computed as if it were on a Cartesian grid, i.e.

$$\underline{W}_F = \frac{\underline{W}_C^+ + \underline{W}_C^-}{2}, \tag{5.1}$$

where $\underline{W}_C^+$ and $\underline{W}_C^-$ are the cell average states on both sides on the cell face. In the following, this reconstruction approach is denoted as *Average* approach.

Alternatively, the face state can be obtained by extrapolation of the cell center values as

$$\underline{W}_{CF}^+ = \underline{W}_C^+ + \Delta\vec{x}^+ \cdot \nabla \underline{W}_C^+, \tag{5.2}$$

where $\underline{W}_{CF}^+$ is the flow state on the cell face extrapolated from the positive cell, $\underline{W}_C^+$ is the flow state at the center of the first cell and $\Delta\vec{x}^+$ is the vector connecting cell center and face center. Doing this for both cells, yields two face states on the cell face that are averaged to obtain the face state

$$\underline{W}_F = \frac{\underline{W}_{CF}^+ + \underline{W}_{CF}^-}{2}. \tag{5.3}$$

This reconstruction approach is denoted as *Extrapolation* approach.

The latter approach requires gradients of the flow state data per cells. Because the grid is non-uniform and non-Cartesian, the simple finite differences cannot be used. The gradients are computed from a second-order accurate least-square approximation,

that utilizes all available face and edge neighbors of the cell. The ansatz for the least-square gradients is

$$\begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta x_1^2 & \Delta x_1 \Delta y_1 & \Delta y_1^2 \\ \Delta x_2 & \Delta y_2 & \Delta x_2^2 & \Delta x_2 \Delta y_2 & \Delta y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \Delta x_n & \Delta y_n & \Delta x_n^2 & \Delta x_n \Delta y_n & \Delta y_n^2 \end{bmatrix} \begin{bmatrix} W_{,x} \\ W_{,x} \\ W_{,xx} \\ W_{,xy} \\ W_{,yy} \end{bmatrix} = \begin{bmatrix} W_1 - W \\ W_2 - W \\ \vdots \\ W_n - W \end{bmatrix}, \tag{5.4}$$

where Δx_i and Δy_i are the relative location of the surrounding points with respect to the point, where the gradient is computed. In this ansatz W is an exemplary conserved quantity and W_i is the value of this quantity at the ith surrounding point. We see that this ansatz only has a solution if five or more surrounding points are used. For more points the system of linear equations is over determined. It is solved with an efficient least-square approach, see Appendix C.

The gradients of the flow field on the cell faces are also computed by a least-square approach. Due to the availability of fewer points on the cell faces, an first-order ansatz is chosen for both reconstruction approaches. This first order ansatz is

$$\begin{bmatrix} \Delta x_1 & \Delta y_1 \\ \Delta x_2 & \Delta y_2 \\ \vdots & \vdots \\ \Delta x_n & \Delta y_n \end{bmatrix} \begin{bmatrix} W_{,x} \\ W_{,x} \end{bmatrix} = \begin{bmatrix} W_1 - W \\ W_2 - W \\ \vdots \\ W_n - W \end{bmatrix}. \tag{5.5}$$

It is solved with the same least-square algorithm as the second-order ansatz. This first order ansatz is also found in [101, Chapter 5.3.4]. This gradient computation is reported to be prone to decoupling. Hence, we use the decoupling correction

$$\nabla \underline{W} = (\nabla \underline{W})_{LS} - \left((\nabla \underline{W})_{LS} \cdot \vec{t} - \frac{\underline{W}^+ - \underline{W}^-}{d} \right) \vec{t} \tag{5.6}$$

with $(\nabla \underline{W})_{LS}$ being the gradient approximation by the least-square ansatz, $\vec{t}$ being the unit vector between the two cell centers and d the distance between the cell centers [101, Chapter 5.4.2]. In this correction the directional derivative is corrected to fit the directional derivative between the cell centers.

5.3.3 Validation of the morph-cell algorithm

In this section three test cases with complex boundaries are considered, to test the performance of the morph-cell algorithm.

The first test considers a channel flow. At the inlet a velocity profile is specified and at the outlet a constant pressure. The solution of this benchmark is the pressure gradient in the flow direction. In order to test the morph-cell algorithm, the channel

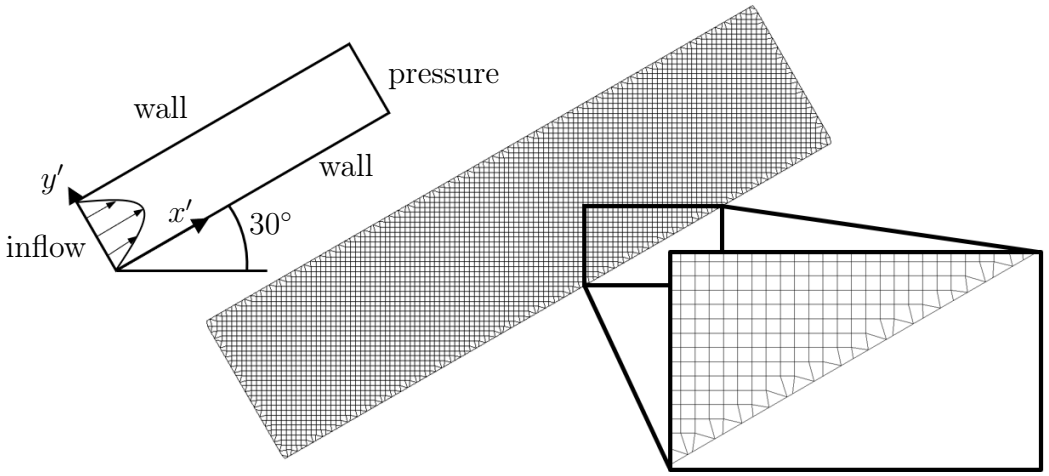


Figure 5.10: Grid for the inclined channel flow simulations

is rotated by 30°, with respect to the Cartesian grid, see Fig. 5.10. The analytical solution for incompressible flow is

$$\frac{\partial p}{\partial x'} = 8\frac{\mu V_{max}}{H^2} \tag{5.7}$$

with x' being the coordinate along the flow direction, V_{max} the maximal velocity in flow direction and H the width of the channel. The simulations are performed with $Ma = 0.001$ and $Re = 1$.

The resulting velocity and pressure fields are shown in Figs. 5.11 and 5.12, respectively. The velocity fields are similar and smooth for both methods and all tested resolutions. The pressure fields are quite different between the two methods. The *Extrapolation* yields smooth pressure fields with a slight decoupling near the morph-cells, while the *Average* yields strong oscillations, that are still visible at higher resolutions.

For a more detailed analysis the error with respect to the analytical solution Eq. (5.7) is plotted in Fig. 5.13 (a). The error for the *Extrapolation* is an order of magnitude smaller that for the *Average*. The *Average* shows only first-order convergence, while the *Extrapolation* converges with second-order, but keeps a constant relative error for higher resolutions. This is an effect of the finite Mach number. Computing the error with respect to the Richardson extrapolation, instead of the analytical solution shows the clear second order of convergence. The deviation between the Richardson extrapolation and the analytical solution is about 0.02%.

The second test case considers the flow around a cylinder as defined by Schäfer et al. [163]. The cylinder has a diameter of $D = 0.1$ and is placed slightly out of the center of a channel with height $H = 0.41$. The grid is shown in Fig. 5.14. It features two refinement levels around the cylinder. The simulation is performed with a $Ma = 0.01$. Reynolds number $Re = 20$ and $Re = 100$ are considered.

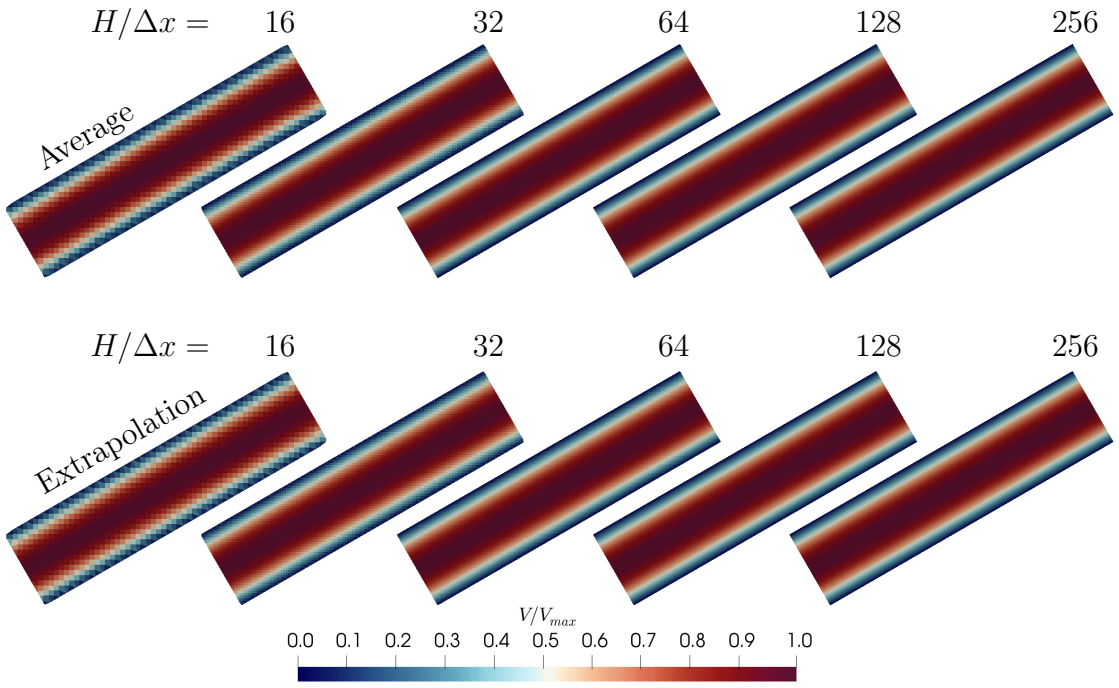


Figure 5.11: Velocity fields for the inclined channel flow simulations

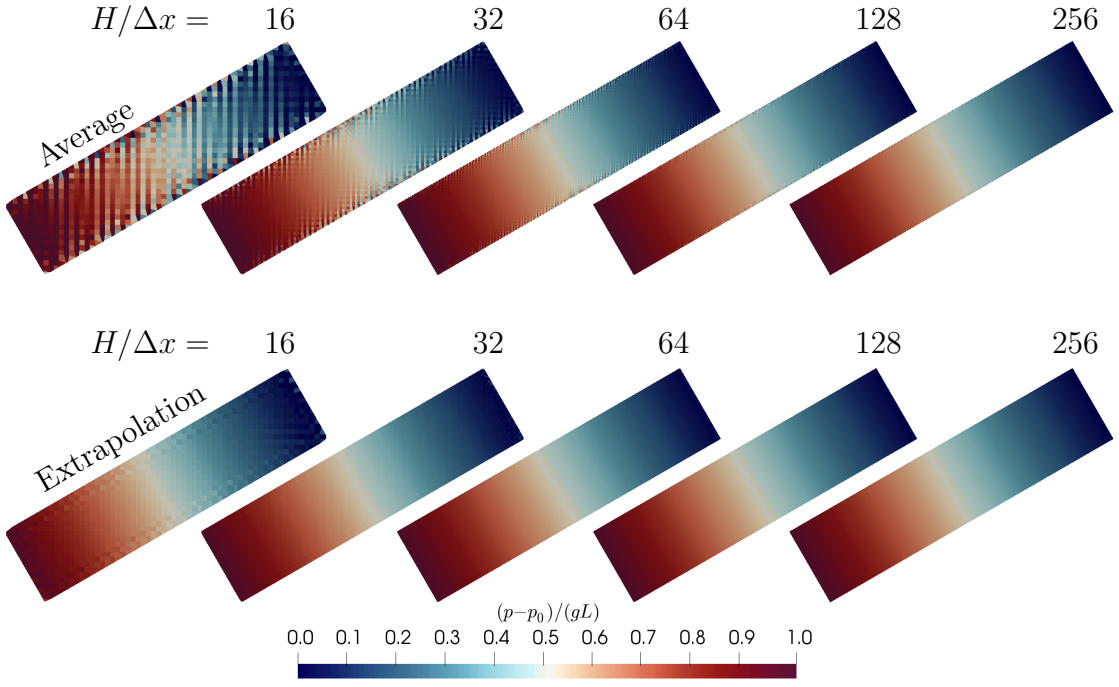


Figure 5.12: Pressure fields for the inclined channel flow simulations

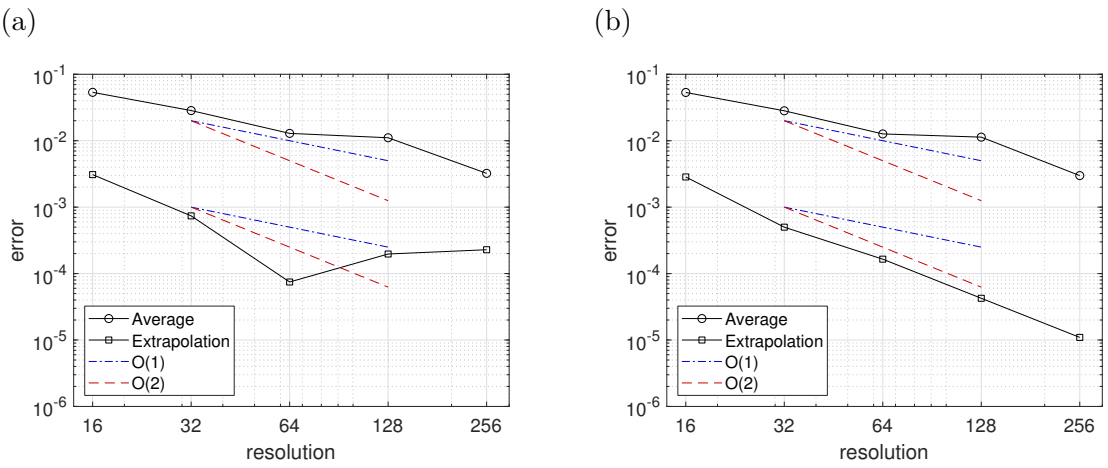


Figure 5.13: Relative pressure error in the channel flow simulations: (a) error with respect to analytical solution, (b) error with respect to Richardson extrapolation.

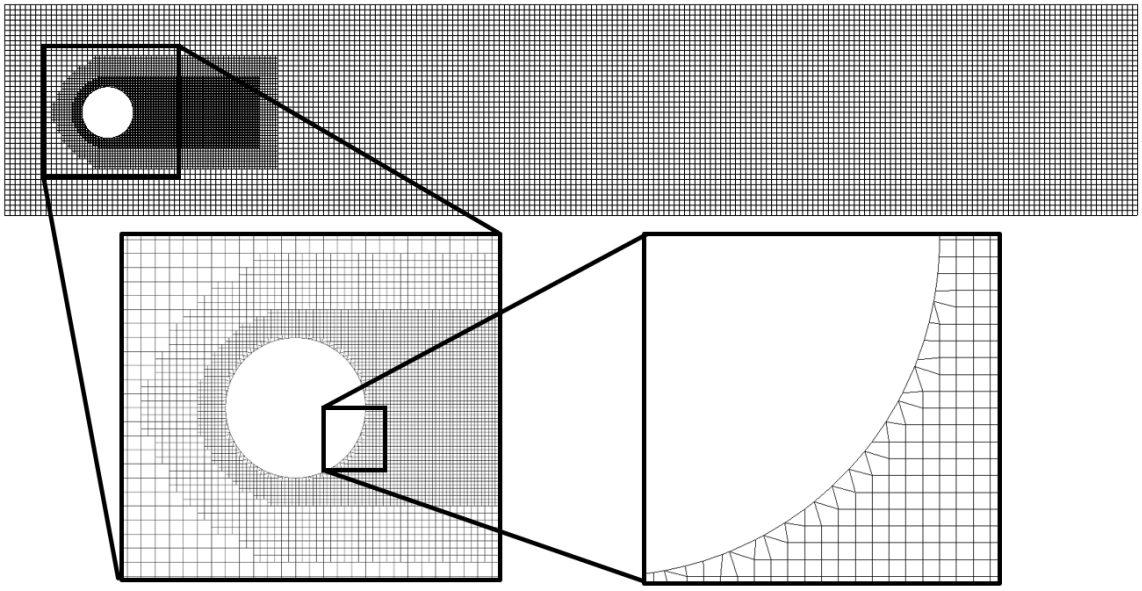


Figure 5.14: Grid for the simulation of flow around a cylinder for the coarsest resolution where $D/\Delta x = 40$

The velocity fields at $Re = 20$ are smooth for both faces state approaches, hence only one is shown in Fig. 5.15. The pressure fields show a similar behavior as observed in the channel flow. The pressure field with the *Average* reconstruction shows strong oscillations that are only terminated at the grid interface. The *Extrapolation* reconstruction shows only a slight decoupling of the pressure.

For validation drag and lift coefficients are compared to the reference of Schäfer et al.

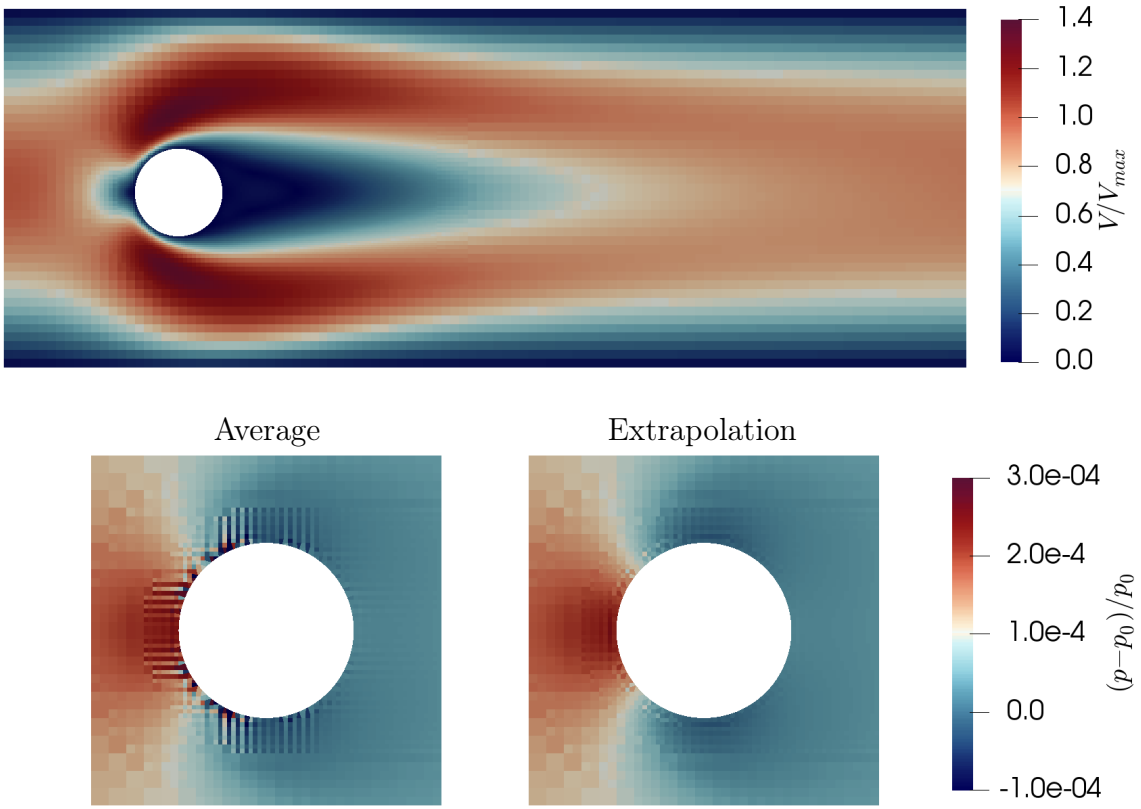


Figure 5.15: Velocity and pressure fields in the simulation of flow around a cylinder at $Re = 20$ for the coarsest resolution where $D/\Delta x = 40$

Table 5.1: Results of flow around cylinder at $Re = 20$

	Average				Extrapolation			
D	c_D	c_L	L_a	Δp	c_D	c_L	L_a	Δp
$40\Delta x$	5.66	0.0110	0.085	0.1139	5.59	0.0107	0.085	0.1072
$80\Delta x$	5.60	0.0108	0.085	0.1167	5.58	0.0106	0.085	0.1147
$160\Delta x$	5.59	0.0107	0.085	0.1172	5.58	0.0106	0.085	0.1166
Ref. [163]	5.58	0.0107	0.0847	0.1174	5.58	0.0107	0.0847	0.1174

[163] in Table 5.1. They are defined as

$$c_D = \frac{2F_x}{\rho \overline{V}^2 D} \qquad \text{and} \qquad c_L = \frac{2F_y}{\rho \overline{V}^2 D}, \tag{5.8}$$

respectively. The forces F_x and F_y are measured as momentum fluxes over the cylinder boundaries. The mean velocity is $\overline{V} = \frac{2}{3}V_{max}$. We find that the drag and lift coefficients are better recovered with the *Extrapolation* approach, especially at low resolution. The recirculation length is recovered in all simulations. Finally, the pressure difference of front and back of the cylinder is more accurate for the *Average* approach. Hence, we find that the *Extrapolation* approach is not superior in every metric.

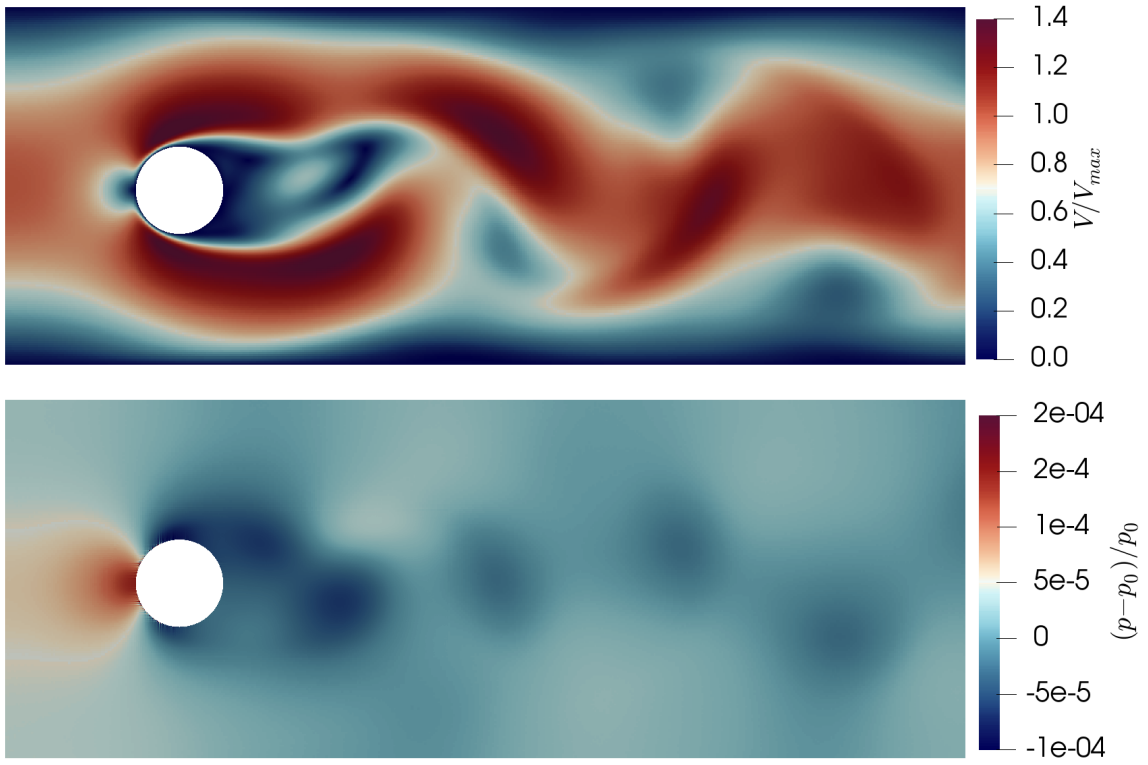


Figure 5.16: Velocity and pressure fields in the simulation of flow around a cylinder at $Re = 100$ with a resolution of $D/\Delta x = 160$ and with the *Average* reconstruction

Table 5.2: Results of flow around cylinder at $Re = 100$

	Average			Extrapolation		
D	$c_{D,max}$	$c_{L,max}$	St	$c_{D,max}$	$c_{L,max}$	St
$160\Delta x$	3.24	0.99	0.30	—	—	—
Reference [163]	3.23	1.00	0.30	3.23	1.00	0.30

This is confirmed in the last test for the morph-cell algorithm, the fow around a cylinder ar $Re = 100$. Apart from the increased Reynolds number, the test is exactly the same as the one considered before. The resulting flow fields obtained with high resolution of $D = 160\Delta x$ are shown in Fig. 5.16. The pressure oscillations in the front of the cylinder are still visible.

For the validation maximal drag and lift coefficients are computed. They match the reference. Also the Strouhal number

$$St = \frac{fD}{\overline{V}} \tag{5.9}$$

with the frequency f of the lift coefficient is recovered correctly. Unfortunately all simulations with the *Extrapolation* reconstruction are unstable.

These test show that the morph-cell algorithm produces boundary conforming grids. It

further shows that the GKS can be extended to run on these non-cartesian grids. The two numerical approaches show differing properties. The *Extrapolation* reconstruction yields better accuracy and convergence rate, while the *Average* reconstruction has superior stability.

6 Validation and application

This chapter is dedicated to the results obtained with VirtualFluidsGKS. For every solver and numerical scheme validation is imperative. The following section shows results of two test cases for two-dimensional natural convection, obtained with the 2D predecessor of VirtualFluidsGKS. These results were previously published by the author of this dissertation [97]. The second section presents four validation cases in three dimensions obtained with VirtualFluidsGKS. The first is a synthetic flow problem for testing of CFD codes. The remaining three validation cases feature comparison with experimental data. The tests range from synthetic turbulence, over turbulent natural convection to the simulation of fire plumes. Finally, two applications of turbulent natural convection and fire simulation are shown.

6.1 Two-dimensional tests

6.1.1 Square cavity with differentially heated walls

Cavity flows are often used as benchmarks due to the simplicity in boundary conditions. The isothermal lid driven cavity was already shown in Section 2.7.5. For natural convection the square cavity with differentially heated wall was introduced by de Vahl Davis [164]. The two-dimensional cavity has four no-slip walls. Top and bottom walls are insulated. The left wall is set to a constant high temperature, while the right wall is kept at low temperature. The temperature difference is assumed to be small, such that the Boussinesq approximation of vanishing density variation can be applied. The flow is then incompressible and the result symmetric.

Here, we consider the compressible extension at large temperature differences, beyond the validity of the Boussinesq approximation. This extension was formulated by Le Quéré et al. [165] and Paillère et al. [166]. The VirtualFluidsGKS results are compared against the contributions of Becker and Braack [167] and Vierendeels et al. [168, 169].

The boundary conditions are similar to the incompressible benchmark. Further, the fluid viscosity is temperature dependent, based on Sutherland's law [170, 168], i.e.

$$\frac{\mu(T)}{\mu^*} = \left(\frac{T}{T^*}\right)^{\frac{3}{2}} \frac{T^* + S}{T + S}, \tag{6.1}$$

where $S = 110.5\,\mathrm{K}$ and $T^* = 273\,\mathrm{K}$. The base viscosity μ^* is computed from one viscosity-temperature pair $\mu_0 = \mu(T_0)$, such that

$$\mu^* = \mu_0 \left(\left(\frac{T_0}{T^*} \right)^{\frac{3}{2}} \frac{T^* + S}{T_0 + S} \right)^{-1}. \tag{6.2}$$

The thermal diffusivity is coupled to the viscosity at any time by a constant Prandtl number $Pr = 0.71$. Further, the number of internal degrees of freedom is chosen as $K = 3$.

The remaining governing non-dimensional numbers for this benchmark are

$$\epsilon = \frac{T_h - T_c}{T_0} = 1.2, \quad Ra = \frac{Pr\ g\ H^3\ \epsilon}{\nu^2} = 10^6 \quad \text{and} \quad Ba = \frac{g\ H}{R\ T_0} = 0.001, \tag{6.3}$$

which are the dimensionless temperature difference with $T_0 = (T_h - T_c)/2$, the Rayleigh number and the Barometric number, respectively. The latter was introduced by Lenz et al. [15] as a supplement for the Mach number for compressible flows at low Mach number. The physical relevance of the Mach number is connected with high speed flows close to and above the speed of sound and quantifies the compressibility in these flow regimes. The definition of the Mach number requires a velocity, which might not exist or might not be known for all flow problems. A static gas under gravitational field and with temperature gradients cannot be quantified by the Mach number. These two physical effects of compressibility are thermal compressibility, which is quantified by the dimensionless temperature difference ϵ, and compressibility due to external forces, which is not characterized by a standard non-dimensional number. Hence, the Barometric number was introduced as the relation of potential energy and internal energy. Low Barometric numbers imply little influence of the gravitational compression on the pressure field. The name was inspired by the Barometric formula, such that $\rho(H) = \rho(0) \exp(-Ba)$. The Barometric number is connected to the Mach number by

$$Ba = \gamma\ Ri\ Ma^2, \tag{6.4}$$

where γ is the isentropic exponent and

$$Ri = \frac{gH}{U^2} \tag{6.5}$$

is the Richardson number. Note that the Richardson number is not defined for $U \to 0$, while the Barometric number is still defined for zero velocity. For this benchmark Ba is chosen small, because the references do not take gravitational compression into account.

For the simulation forcing scheme (1) was used. The grid is shown on the left of Fig. 6.1. Two refinement steps towards the walls are used, to resolve the thermal and viscous boundary layers.

Stream lines and temperature contours are also shown in Fig. 6.1. For the validation, Nusselt numbers

$$Nu = \frac{q}{\kappa_0 \frac{T_h - T_c}{L}} \tag{6.6}$$

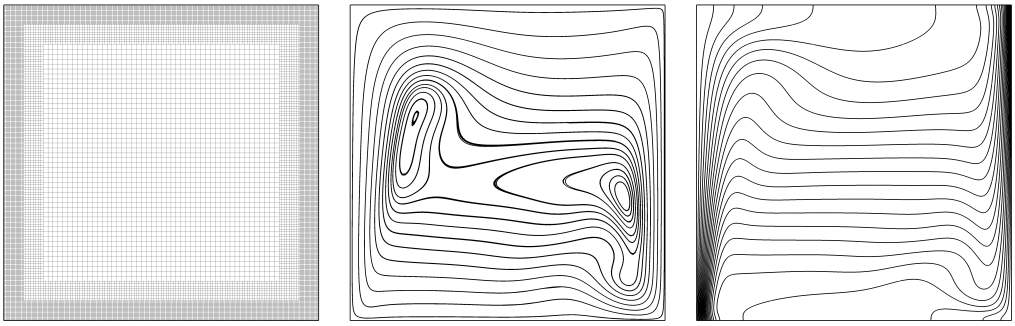


Figure 6.1: 2D square cavity: grid, streamlines and temperature iso-contours. Figure reused from [97].

Table 6.1: Empirical convergence study for the heat transfer in the square cavity with differentially heated sides. Table reused from [97].

Background Resolution	Nu	Values used for Richardson Extrapolation			
64×64	8.681880	×	×		
96×96	8.685609	×			
128×128	8.686268		×	×	×
144×144	8.686388	×			
192×192	8.686541			×	
256×256	8.686609		×		×
288×288	8.686626			×	
512×512	8.686669				×
$\widehat{Nu}$		8.686594	8.686637	8.686663	8.686682
α		3.9	3.7	2.9	2.5
Vierendeels, 2001, [168]	8.686585				
Becker, 2002, [167]	8.686609				

are computed, where q is the wall heat flux, κ_0 is the thermal conductivity at mean temperature and L is the cavity width. The reference value is $Nu = 8.6866$ on both walls [165]. Nusselt numbers for the simulation on the quadtree grid are listed in Table 6.1. The Richardson extrapolation of the Nusselt number is

$$\widehat{Nu} = \frac{Nu_2^2 - Nu_1 Nu_3}{2Nu_2 - Nu_1 - Nu_3}, \tag{6.7}$$

where Nu_1, Nu_2 and Nu_3 are Nusselt numbers of coarse, medium and fine simulations, respectively. All extrapolated Nusselt numbers are similar to the reference values when rounded to four decimals. Formally, GKS and the implemented finite volume method have second order of accuracy $\alpha = 2$. The numerical order of convergence was computed by

$$\alpha = -\ln\left(\frac{Nu_3 - Nu_2}{Nu_2 - Nu_1}\right) / \ln(r), \tag{6.8}$$

where r denotes the refinement ration between the grids, i.e. the decrease in cell size from coarse to medium and from medium to fine. Therein, the error is assumed to

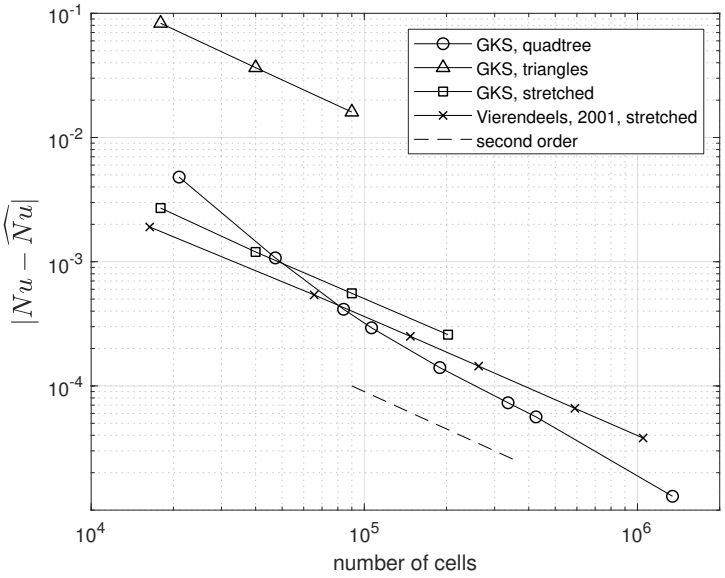


Figure 6.2: 2D square cavity: Nusselt numbers. Figure reproduced from [15, 97].

behave as $Nu_k = \widehat{Nu} + C(\Delta x_0 r^{-k})^\alpha$. The numerical order of convergence is above the expected value of two. Thus, the simulations are still pre-asymptotic.

In order to compare the quadtree-type grid with other grid layouts, it is compared to an early GKS implementation [15] that uses arbitrary cells. The difference between Nusselt number and the best guess, i.e. the Richardson extrapolation of the each method, is shown in Fig. 6.2. The simulations on uniform triangles and on stretched Cartesian meshes both converge with close to second order, same as the reference of Vierendeels et al. [168]. At low resolutions, the quadtree-type converges fast, but also goes towards the second order.

This test case shows that GKS on quadtree-type meshes is able to predict compressible natural convection. It further shows that the quadtree-type grid makes better usage of the degrees of freedom than the triangular cells and is comparable to the stretched Cartesian grids.

6.1.2 Rayleigh-Bénard at high Rayleigh number

The Rayleigh number of the simulation in the prior section was comparably small, when considering that fires can easily have Rayleigh numbers of $Ra = 10^{13}$. Unfortunately, no benchmarks exist for high Rayleigh number natural convection at large temperature differences. Therefore, this section presents high Rayleigh number natural convection at low temperature differences, i.e. in the Boussinesq limit. Here, we present the results for the Rayleigh-Bénard benchmark of van der Poel et al. [171] and Bao et al. [172]. The setup is square cavity with no-slip walls. Left and right walls are insulated and top and bottom walls have low and high temperature, respectively. The Rayleigh number is varied between 10^9 and 10^{12} and the Prandtl number is set

Table 6.2: Details of the meshes for the Rayleigh-Bénard convection in the Boussinesq limit. Table reused from [97].

Ra	background resolution	refinements	$\Delta x/H$
10^9	512×512	2	$4.88 \cdot 10^{-4}$
10^{10}	512×512	3	$2.44 \cdot 10^{-4}$
10^{11}	1024×1024	2	$2.44 \cdot 10^{-4}$
10^{12}	1024×1024	3	$1.22 \cdot 10^{-4}$

to 4.3, which is a value for water. For this setup it is found that the Nusselt numbers follow the model [172]

$$Nu_m = 0.1 Ra^{0.3}. \tag{6.9}$$

In terms of non-dimensional numbers the Boussinesq limit is characterized by $Ba \to 0$, $\epsilon \to 0$ and $Ma \to 0$. In terms of physics, it is characterized by vanishing density variations. Based on the compressible GKS solver with scalar transport two options exist to reach this limit. First, the non-dimensional numbers can be set to very low values. This approach is denoted as CS, for compressible solver. Strongly reducing the Barometric number increases the computational demand, because of smaller time steps. For the CS approach $\epsilon = 0.01$ and $Ba = 0.001$ are chosen. The second approach is to separate temperature field and flow field. To this end we introduce the pseudo temperature S, which is traced as a scalar field. In order to preserve the symmetry, the mean value is $S_0 = 0$. The GKS temperature remains T. It will only show small deviations from a chosen T_0, which are due to pressure variations and viscous heating. The value of T_0 is chosen based on $Ma = 0.1$, which will be sufficiently small to reduce compressibility errors. Instead of a global forcing, a local forcing

$$g_l(\vec{x}) = \beta \left(S_0 - S(\vec{x})\right) g \tag{6.10}$$

with β being the thermal expansion coefficient $\beta = \epsilon/\Delta S$ is used. The corresponding Rayleigh number is

$$Ra = \frac{Pr\ g\ L^3\ \beta\ \Delta S}{\nu^2}. \tag{6.11}$$

The grids for these simulations feature multiple refinement steps towards top and bottom walls. The first refinement extends over a distance of $L/16$ from the wall, while the second and third refinement quarter that distance, i.e. $L/64$ and $L/256$. The background resolution and the number of refinement steps used for the grids for the four Rayleigh numbers are given in Table 6.2. The PS approach was used to simulate $Ra = 10^9$, 10^{10}, 10^{11} and 10^{12}, while the CS approach was only used for $Ra = 10^9$ and 10^{10} due to excessive time demands.

The resulting instantaneous pseudo temperature fields are shown in Fig. 6.3. Plumes of hot and cold fluid are visible transporting the heat from bottom to top. With increasing Rayleigh number the plumes get thinner and the thermal boundary layers at top and bottom are nearly invisible. When the plumes detach from the wall, large vortices are visible. The heat transport is highly turbulent. For $Ra = 10^{11}$ and 10^{12}

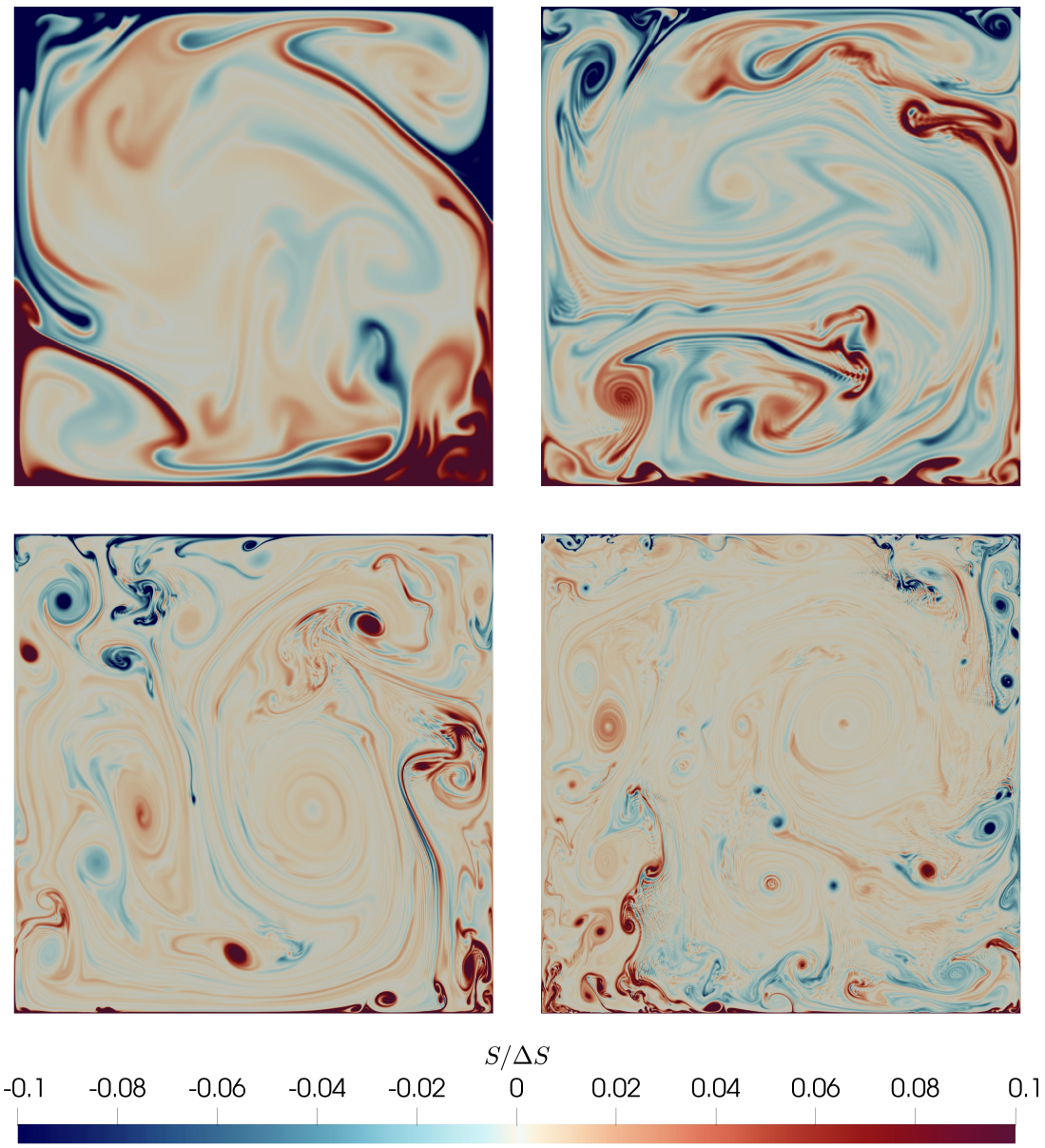


Figure 6.3: Pseudo temperature for Rayleigh-Bénard convection at high Rayleigh numbers. The Rayleigh numbers are $Ra = 10^9$, 10^{10}, 10^{11} and 10^{12} for top-left, top-right, bottom-left and bottom-right, respectively. The results were obtained with the PS approach. [97]

the temperature fields show streaks, which might be attributed to insufficient temporal resolution [172]. The finite Mach number might also introduce these streaks.

Due to the turbulent nature of the heat transport in the cavity, the Nusselt number is not constant over time. Hence, it must be averaged for validation. In order to define a time scale for turbulence averaging, we use the free fall time

$$t^* = \frac{L}{\sqrt{g\beta\Delta SL}} = \frac{L}{\sqrt{g\epsilon L}} \tag{6.12}$$

Table 6.3: Nusselt numbers for high Rayleigh number Rayleigh-Bénard convection in a square cavity. The solvers are the passive scalar solver (PS) and the compressible solver (CS). The results are compared to the model $Nu_m = 0.1Ra^{0.3}$ and the reference simulation from Bao et al. [172]. Table reused from [97].

Ra	Solver	Avg. Int.	Nu_{ref} [172]	Nu_m	Nu_{GKS}	$\frac{Nu - Nu_m}{Nu_m}$
10^9	PS	500 t^*	51.57	50.12	51.03	2.1%
10^{10}	PS	500 t^*	100.20	100.00	101.99	2.0%
10^{11}	PS	500 t^*	198.53	199.53	196.20	−1.7%
10^{12}	PS	235 t^*	380.59	398.11	383.84	−3.6%
10^9	CS	200 t^*	51.57		49.42	−1.4%
10^{10}	CS	100 t^*	100.20		91.94	−8.1%

introduced by Bao et al. [172].

The results for the Nusselt number including the averaging intervals are listed in Table 6.3. All Nusselt numbers are close to the reference computations, as well as the model, see Eq. (6.9). For the PS approach the difference is well below 5 %. At the lower Rayleigh number of $Ra = 10^9$ the CS approach predicts a Nusselt number that deviates from the model by only 1.4 %. For the higher Rayleigh number of 10^{10} the deviation is already 8.1 %. The larger deviation might be due to the substantially shorter average time.

This benchmark shows that GKS can be used for turbulent natural convection in the Boussinesq limit. Including the prior section that validated GKS for compressible natural convection, we argue that the present GKS is valid for the simulation of turbulent natural convection also at large temperature differences.

6.1.3 Demonstration of compressible two-dimensional turbulent Rayleigh-Bénard convection at high Rayleigh number

In this last section on two-dimensional simulations, we demonstrate the simulation of compressible natural convection at $Ra = 10^{13}$ at the large temperature difference of $\epsilon = 1.2$. The Prandtl number is 1.0. If one converts these parameters to real world units, with choosing a cold temperature of $T_c = 25\,°\mathrm{C}$, the viscosity of air as $\nu = 1.7 \cdot 10^{-5}\,\mathrm{m^2\,s^{-1}}$ and $g = 9.81\,\mathrm{m\,s^{-2}}$, then the resulting length scale is $L \approx 6\,\mathrm{m}$ and the upper temperature is $T_h \approx 1200\,°\mathrm{C}$. These parameters approximate the conditions that appear in fire.

The lateral boundary conditions are periodic. The domain is split into about 15 million cells with four refinement levels, such that the cells on the wall have a size of about $H/35,000$. The initial condition is the average temperature T_0. The temporal evolution of the temperature fields at $t = 5t^*$, $15t^*$, $30t^*$ and $100t^*$ are shown in

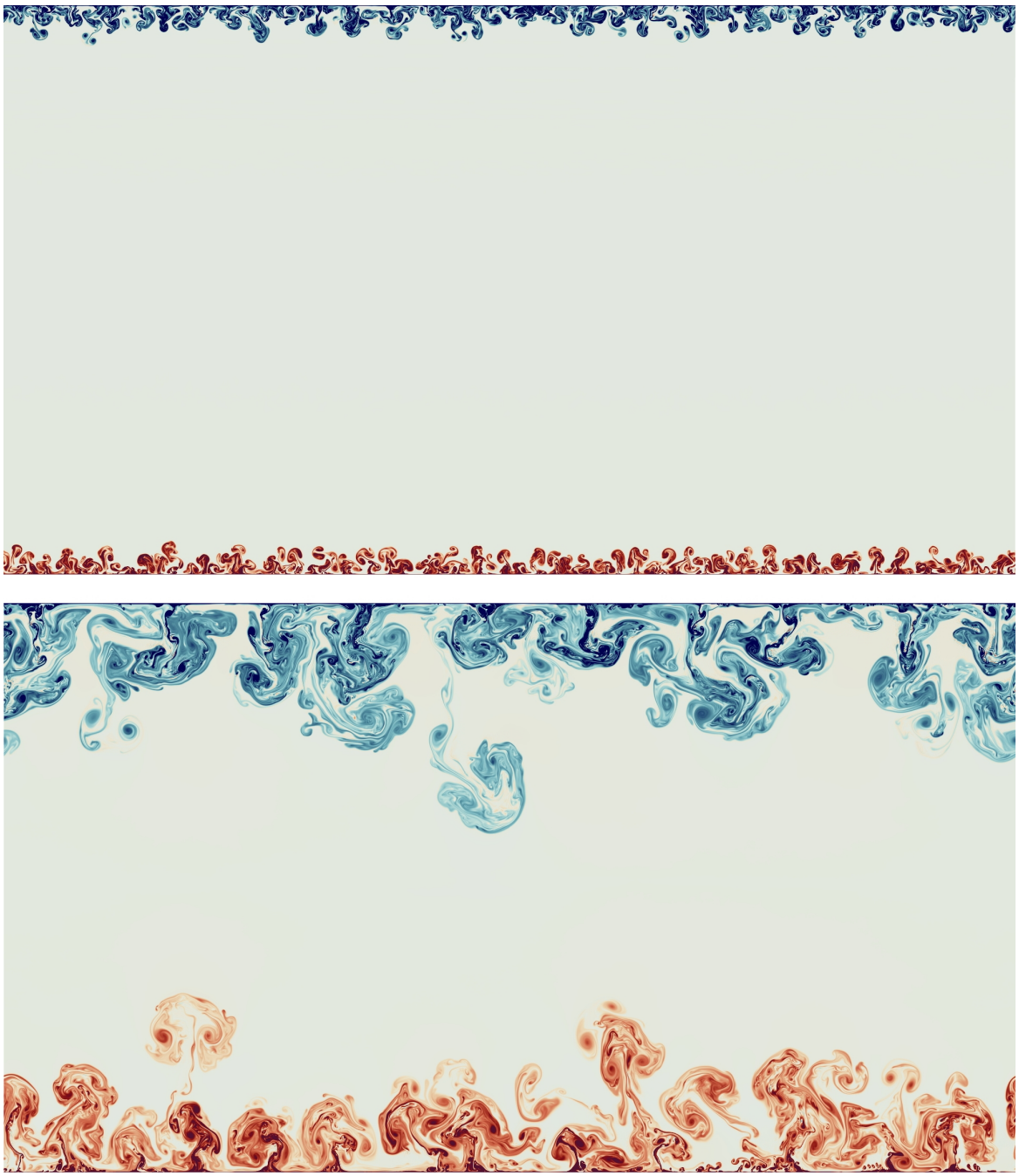


Figure 6.4: Rayleigh-Bénard convection at high Rayleigh number $Ra = 10^{13}$ and large temperature difference $\epsilon = 1.2$. The two frames are at $t = 5t^*$ and $15t^*$. The simulation is also available as video under `https://youtu.be/BJKiuwpdprQ`. Figure reproduced from [97].

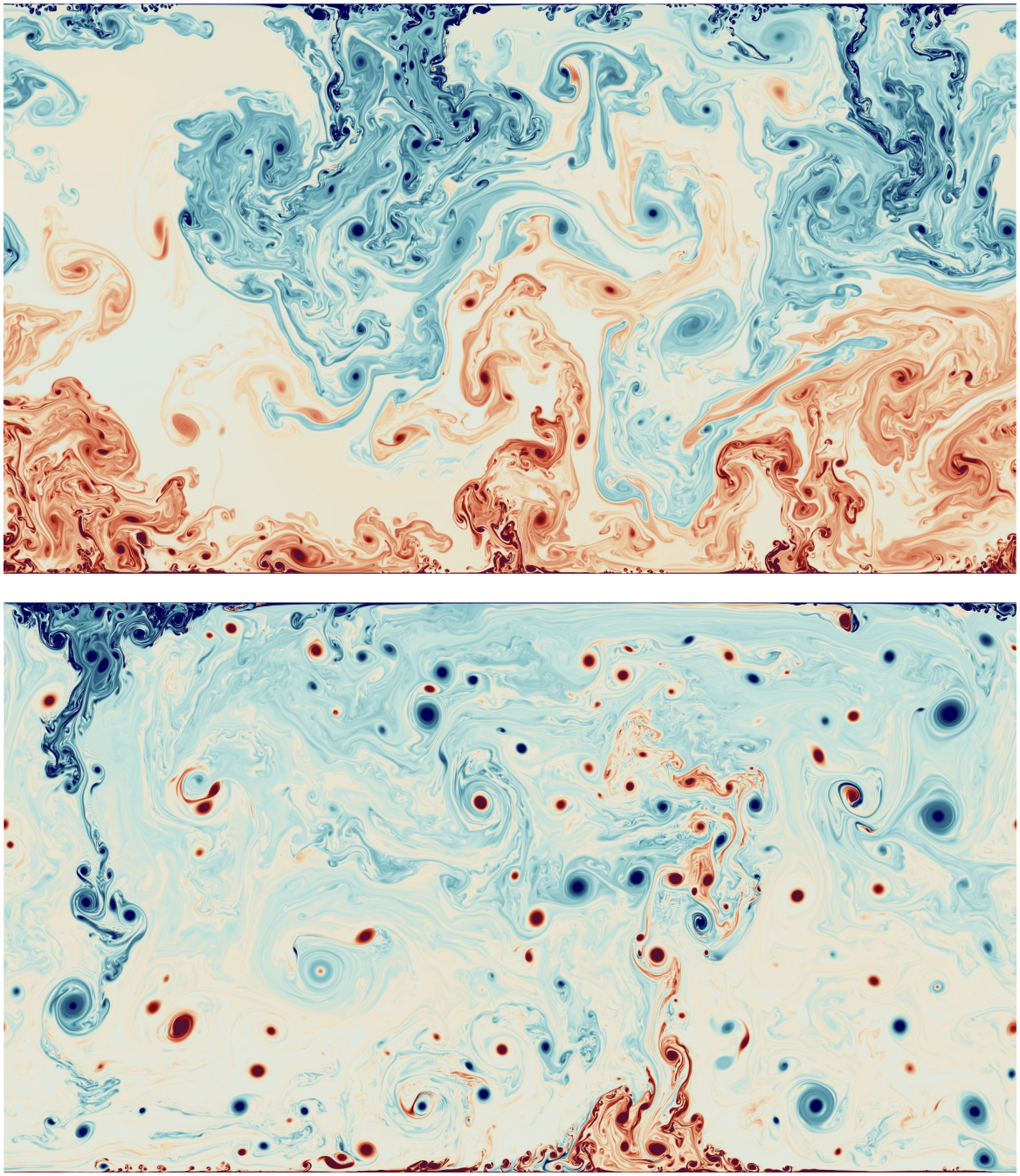


Figure 6.5: Rayleigh-Bénard convection at high Rayleigh number $Ra = 10^{13}$ and large temperature difference $\epsilon = 1.2$. The two frames are at $30t^*$ and $100t^*$. The simulation is also available as video under `https://youtu.be/BJKiuwpdprQ`. Figure reproduced from [97].

Figs. 6.4 and 6.5, where t^* is defined in Eq. (6.12). At the beginning, small plumes eject from both top and bottom walls. The scale of these first plume is solely determined by the grid resolution. Only later they merge to larger plumes. At $t = 30t^*$ the hot and cold regions begin to mix. In the final state, two major plumes, one for hot fluid and one for cold fluid, are found. This is an effect of two-dimensional turbulence, where energy is transported from small to large scales [173], such that the final state shows two large scale vortices, one rotating clockwise and one rotating anti clockwise.

This simulation was also turned into a video that is available on YouTube under

`https://youtu.be/BJKiuwpdprQ`.

Even though this section provides no new validation, it demonstrates the capability of GKS implementations on GPUs.

6.2 Three-dimensional tests

6.2.1 Turbulent decay of the Taylor-Green vortex

With this first three-dimensional test case we aim at showing the validity of VirtualFluidsGKS for the direct simulation of three-dimensional turbulence. These simulations are also part of a paper that is submitted for publication [174]. Therein GKS is compared to different LBM variants.

The Taylor-Green vortex describes a synthetic flow field that satisfies the time dependent Navier-Stokes equations [175]. The two-dimensional Taylor-Green vortex is stable and preserves its shape over time. Only dissipation takes place, such that the velocity magnitude is reduced with time. The vorticity in two dimensions is a scalar that cannot be generate by the non-linear term in Navier-Stokes equations [176]. Hence, the vorticity remains the same. In three dimensions vorticity is generated and vortex stretching occurs. The energy is then transported from large scales to small scales. Due to this cascading process, the flow transitions into turbulence.

The initial condition for the Taylor-Green vortex can be specified exactly, such that the solution can be compared between different methods and solvers. Here, we follow the setup of Wang et al. [177], which was also solved by Jacobs et al. [178] and Foti and Duraisamy [179]. The flow domain is a periodic cube with $\vec{x} \in (-\pi L, \pi L)^3$. The Reynolds number based on the reference length L is

$$Re = \frac{\rho_0 U_0 L}{\mu} = 1600. \tag{6.13}$$

In the incompressible limit the Reynolds number completely defines the flow problem. GKS as a compressible method requires the definition of further parameters. The

simulations in [178] also used a compressible solver, such that their parameters are chosen here. These parameters are

$$Ma = 0.1\ , \qquad Pr = 0.71\ , \qquad \text{and} \qquad \gamma = 1.4. \tag{6.14}$$

The initial condition is given by Wang et al. [177] as

$$\begin{aligned} U &= U_0 \sin(x/L)\cos(y/L)\cos(z/L) \\ V &= -U_0 \cos(x/L)\sin(y/L)\cos(z/L) \\ W &= 0 \\ p &= p_0 + \frac{\rho_0 U_0^2}{16}\big(\cos(2x/L) + \cos(2y/L)\big)\big(2 + \cos(2z/L)\big). \end{aligned} \tag{6.15}$$

The evolution of the flow field is observed for $20t^*$, where $t^* = L/U_0$ is a reference time. In order to analyze the behavior of the solver, three resolutions in space and three resolutions in time are computed, making a total of nine simulations. The spatial grids contain 64^3, 128^3 and 256^3 cells. For the temporal resolution we define a base time step $\Delta t_0 = t^*/250$ for a spatial resolution of 64^3. For each spatial resolution three temporal resolutions are computed, where the time steps are always halved. The CFL number is kept constant between the spatial resolutions, such that the time steps are $\{\Delta t_0, \frac{1}{2}\Delta t_0, \frac{1}{4}\Delta t_0\}$ for 64^3, $\{\frac{1}{2}\Delta t_0, \frac{1}{4}\Delta t_0, \frac{1}{8}\Delta t_0\}$ for 128^3 and $\{\frac{1}{4}\Delta t_0, \frac{1}{8}\Delta t_0, \frac{1}{16}\Delta t_0\}$ for 256^3.

The results of the simulations are shown in Fig. 6.6 in terms of iso-contours of the z-component of vorticity. For a qualitative comparison, the visualization is tuned to resemble the results of Jacobs et al. [178], which are also shown in the figure. At $t = 10t^*$ a good agreement of the individual structures is observed, while at $t = 20t^*$ individual structures cannot be correlated between the two simulations anymore. This is due to the chaotic nature of turbulence, where a small difference at some point in time yields a completely different solutions later on. The scale of the turbulent structures and the overall distribution of vorticity are comparable, though.

In order to quantify the quality of the solution, the integral kinetic energy E_{kin} and the integral enstrophy $\mathcal{E}$ are computed. Time series for both quantities are reported in the references [177, 178]. The kinetic energy is calculated by

$$E_{kin} = \frac{1}{2\rho_0 V}\int_V \rho\vec{U}\cdot\vec{U}dV, \tag{6.16}$$

where $V = (2\pi L)^3$ is the periodic domain. Further, ρ is the instantaneous local density and ρ_0 is the reference, i.e. mean, density.

The integral enstrophy $\mathcal{E}$ is a kinematic measure for the dissipation of kinetic energy. It hence describes only the motion, without considering forces. It can be computed in

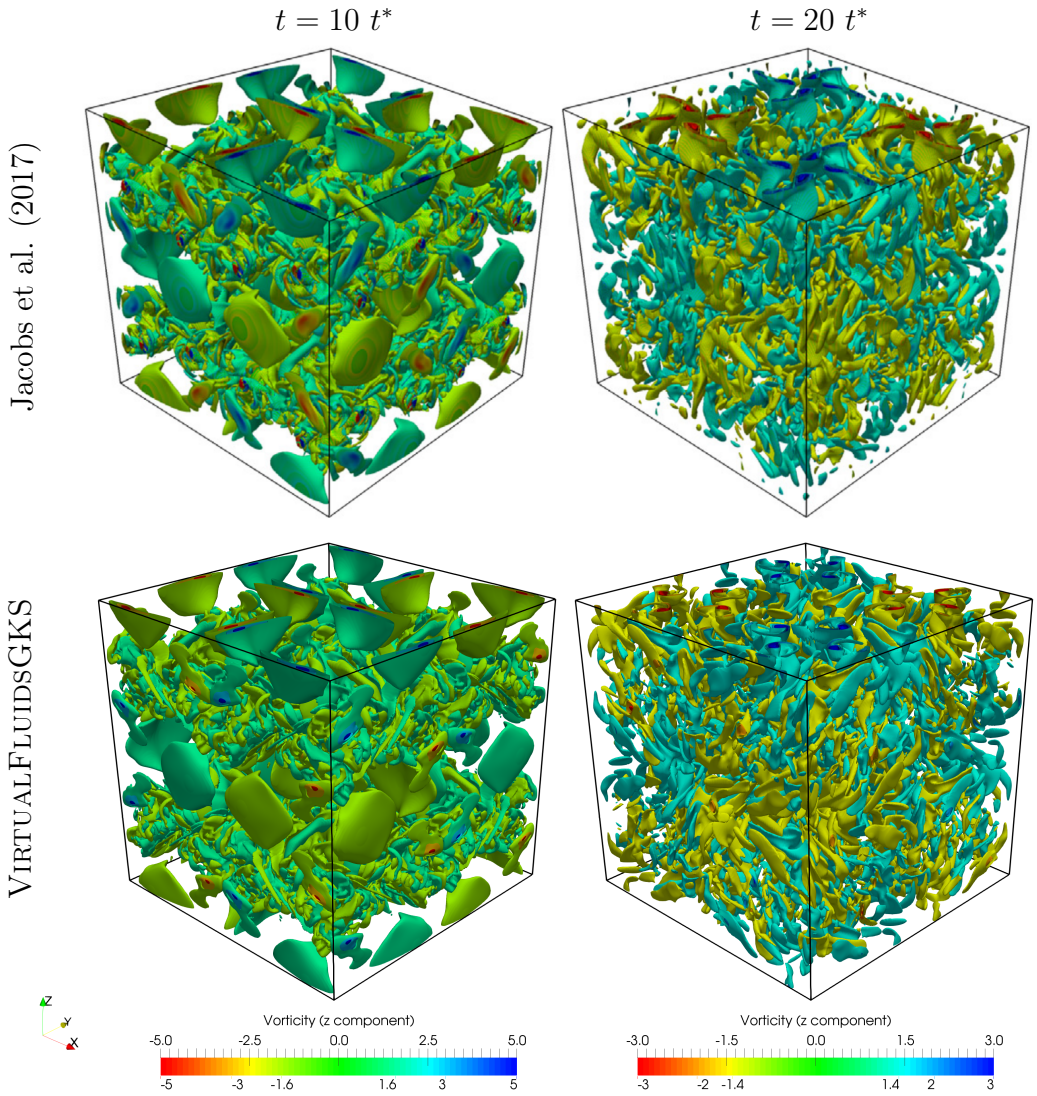


Figure 6.6: Iso-contours of z-vorticity for decaying Taylor-Green vortex at ten and twenty reference times $t^* = L/U_0$. The results of Jacobs et al. [178] are copied from their paper. The annotations below the color bars denote the vorticity values of the iso-contours.

two different ways, which should yield identical results for incompressible flow [177]. First, the enstrophy is proportional to the time derivative of the kinetic energy [104], i.e.

$$\mathcal{E} = -\frac{1}{2\nu}\epsilon = -\frac{1}{2\nu}\frac{\partial E_{kin}}{\partial t}, \tag{6.17}$$

where ϵ is the dissipation rate. Second, the enstrophy can be computed locally from the vorticity $\vec{\omega} = \nabla \times \vec{U}$ as a kinematic quantity [177, 178]. The integral enstrophy is then given as

$$\mathcal{E} = \frac{1}{2\rho_0 V}\int_V \rho\vec{\omega}\cdot\vec{\omega}dV. \tag{6.18}$$

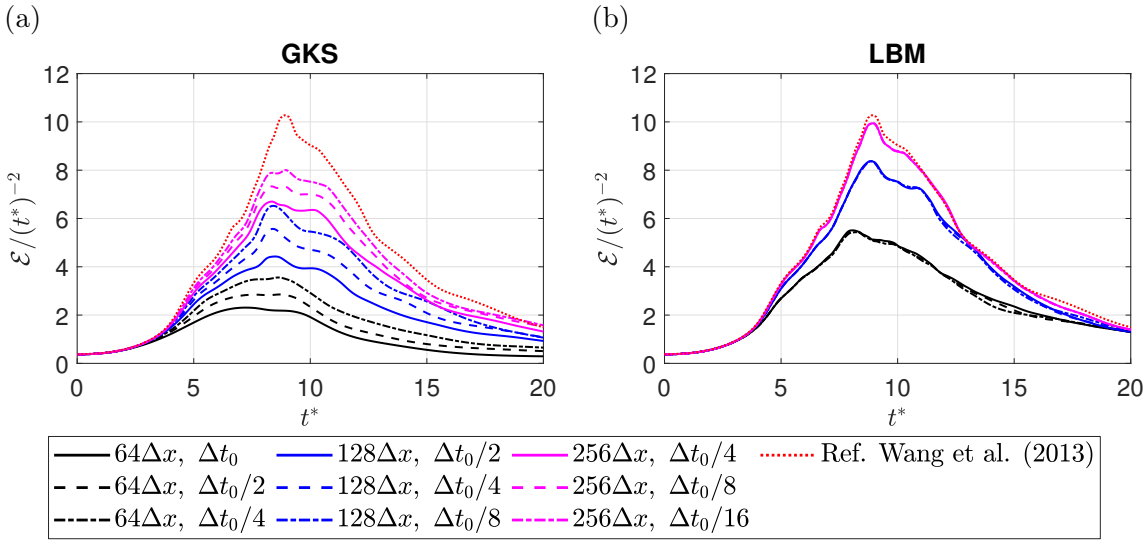


Figure 6.7: Integral kinetic energy and enstrophy for Taylor-Green vortex at $Re = 1600$, compared to spectral result by Wang et al. [177]. The LBM solution is obtained with fourth-order convergent LBM [180, 181, 182].

The first definition stems from dynamics, i.e. the dissipation of energy, while the second stems from kinematics. The integral enstrophy by the second definition can be used as a measure for small scale structures in the flow. For spatially resolved flow simulations it is, hence, a measure for how many small scale structures are present in the solution.

Fig. 6.7 (a) shows the time series of E_{kin} and $\mathcal{E}$ together with the spectral solution published by Wang et al. [177]. The enstrophy is computed from the local vorticity. It shows a large variation for the different simulations. Increasing spatial resolution substantially increases the enstrophy. On a coarse grid small scales cannot be represented. Hence, the large local vorticity of the small scales of the turbulence is missing in the integral enstrophy. Even at the highest resolution of 256^3 cells the difference to the reference is large. Without investigating other results this could easily be attributed to the spatial resolution. Comparing the GKS results to results obtained with a fourth-order convergent LBM [180, 181, 182] reveals that local methods can obtain much more enstrophy on the same grid, see Fig. 6.7 (b). For GKS, the lower enstrophy stems from the finite volume discretization, which introduces numerical viscosity and, hence, dissipates the smallest scales. We further recognize, that the enstrophy in GKS depends strongly on the time step length. This behavior is not observed for the fourth-order convergent LBM. For the finest grid with 256^3 cells the enstrophy converges for time steps $\Delta t = \Delta t_0/256$, see Fig. D.4 in the appendix. Further, we found that the computed enstrophy depends strongly on the accuracy of the finite differences used to compute the vorticity. Fig. D.5 in the appendix compares results obtained with second-order and eights-order accurate central finite differences. The enstrophy computed with the higher order finite difference is substantially higher and, hence, closer to the reference. For the results above, eights-order finite differences were

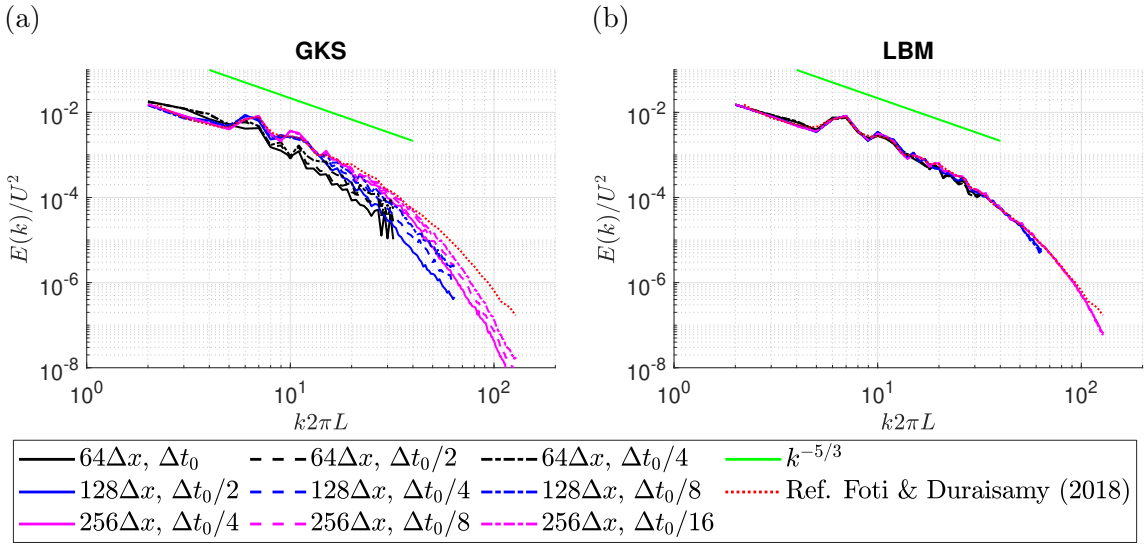


Figure 6.8: Energy density spectrum for Taylor Green vortex, compared to the result by Foti and Duraisamy [179]. The LBM solution is obtained with fourth-order convergent LBM [180, 181, 182].

used.

The kinetic energy time series in Fig. D.3 in the appendix show good agreement with the reference for the two finer resolutions. For the coarse resolution initially too much energy is dissipated while later the missing small scale structures reduce the dissipation, such that energy decreases less than in the reference.

In addition to the time series we investigate the turbulent energy spectrum at $t = 10t^*$. For turbulent flow the shape of the turbulent kinetic energy spectrum over length scales is known. It has an initial region, where turbulent kinetic energy is produced on large scales, a cascading region, where energy is transported from large scales to small scales, and at low scales a dissipative region, where turbulent kinetic energy is dissipated. The Reynolds number of 1600 is relatively low for turbulent flow. The enstrophy shows that the small scale structures vanish fast. The time $t = 10t^*$ is chosen because it is close to the maximal enstrophy, i.e. the maximal turbulence. In the cascading region the slope of the energy spectrum is $E \propto k^{-5/3}$. Due to the low Reynolds number this slope is only observed for few length scales. The beginning of the dissipative region can be quantified by the Kolmogorov length

$$\eta = \left(\frac{\nu^3}{\epsilon}\right)^{\frac{1}{4}} = \left(\frac{\nu^2}{2\mathcal{E}}\right)^{\frac{1}{4}} = \left(\frac{L^4}{2(Re\ t^*)^2\mathcal{E}}\right)^{\frac{1}{4}} \approx 0.012L, \tag{6.19}$$

where the enstrophy is taken from the reference in Fig. 6.7 as $\mathcal{E} \approx 9(t^*)^{-2}$. The Kolmogorov length translates to grid spacings as $0.12\Delta x$, $0.24\Delta x$ and $0.49\Delta x$ for 64^3, 128^3 and 256^3, respectively. Hence, none of the resolutions is a true DNS, because the Kolmogorov length cannot be represented on the grid.

The energy spectra are computed with a MATLAB code provided on GitHub by Felix Dietzsch [183]. Therein, the energy spectra are computed by integrating shells of equal wave numbers, where high wave numbers correspond to small length scales. The energy spectrum obtained by GKS is found in Fig. 6.8 (a), where also a reference spectrum by Foti and Duraisamy [179] is shown. For the lower wave number, i.e. larger length scales, the spectra correspond well. Due to the numerical dissipation higher wave numbers are damped, such that they show less energy. Increasing the spatial and/or temporal resolution improves this behavior. The result of the fourth order convergent LBM is shown in Fig. 6.8 (b). Its resemblance with the reference is remarkable. The spectra of the coarser grid just drop very slightly for the higher wave numbers.

In this section, the direct simulation of turbulence was investigated. VirtualFluidsGKS can in general obtain correct solutions for the turbulent decay in the Taylor-Green vortex. Compared with spectral reference solutions and the fourth-order convergent LBM it shows deficiencies, which are to be expected of a second-order method compared to a fourth-order method.

6.2.2 Turbulent convection in a square cavity

Validation of three dimensional thermal compressible natural convection at large temperature differences is difficult, due to the absence of experimental data. At low temperature differences well described experimental data is available though, and will, hence, be used for validation. Detailed experiments of a three dimensional air filled square cavity with differentially heated walls were performed by Tian and Karayiannis [184, 185] in 2000. These experiments were repeated numerically by Salinas-Vázquez et al. [186] in 2011 with a compressible LES solver.

The experimental cavity had a size of $L_x \times L_y \times L_z = 0.75 \times 1.5 \times 0.75\,\mathrm{m}^3$. The hot and cold walls normal to the x-direction were kept at constant temperature by continuos water flow between steel plates. The top and bottom walls (normal to the z-direction) also feature steel plates, such that they are conductive. This is a difference to the synthetic 2D tests in Section 6.1.1, where insulated walls were used for top and bottom. The flow field in the experiment is solely evaluated at the center $x-z-$plane. The extend of the y-direction is larger to reduce wall effects on the center plane. In y-direction the cavity is closed with glass panels in order to allow observation [184].

Hot and cold walls are kept at temperatures $T_h = 323\,\mathrm{K}$ and $T_c = 283\,\mathrm{K}$, respectively. The average temperature is equal to the room temperature of $T_0 = 303\,\mathrm{K}$. These temperatures yield a non-dimensional temperature difference of $\epsilon = \Delta T/T_0 = 0.132$. The medium in the cavity is air. Tian and Karayiannis [184] report a Rayleigh number of

$$Ra = \frac{Pr\ g\ H^3\ \epsilon}{\nu^2} = 1.58 \cdot 10^9. \tag{6.20}$$

Salinas-Vázquez et al. [186] use a cubic cavity with side length L and a non-uniform orthogonal grid that is stretched in x- and z-directions to increase boundary layer resolution. The wall normal resolution is $\Delta x/L = 3 \cdot 10^{-4}$. Periodic boundaries are used on the y-normal walls to mimic a larger cavity. Hot and cold walls feature isothermal boundary conditions. For top and bottom walls, Tian and Karayiannis [184] report the temperature profiles. A cubic fit of the reported temperatures is used to set boundary values. The exact interpolation used in the reference simulation is not reported. Fig. 6.9 shows the measured boundary temperatures and the cubic fit of the data.

The present GKS simulations use a similar setup as described by Salinas-Vázquez et al. [186]. The grid is a uniform Cartesian grid with octree based refinement. The background grid has a resolution of 128^3 cells with a cell size of $\Delta x/L = 7.8{\cdot}10^{-3}$. The walls are refined with three additional levels, resulting in a cell size of $\Delta x/L = 9.8 \cdot 10^{-4}$, and, hence, having a slightly lower wall normal resolution then the simulation of Salinas-Vázquez et al. [186].

The chosen parameters for this simulation are a Barometric number of $Ba = 0.1$, Prandtl number of $Pr = 0.71$ and $K = 2$. Reference temperature is set to fulfill the Barometric number and viscosity is set to fulfill the Rayleigh number. All free

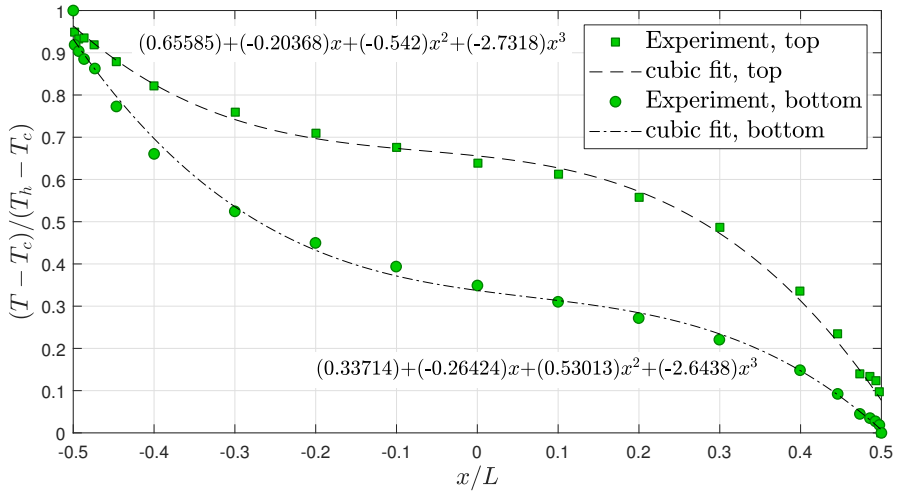


Figure 6.9: 3D Square cavity: top and bottom boundary temperatures. In the experiment top and bottom walls are highly conductive. The convective heat transport in the cavity yields non-linear temperature curves at top and bottom walls. A cubic fit is applied to the experimental data from Tian and Karayiannis [184] and then used to prescribe the boundary temperature as done in the reference simulation [186].

parameters are chosen as unity. For temperature dependent viscosity Sutherland's law (see Eq. (6.1)) is used. As turbulence model the static Smagorinsky model with $C_S = 0.2$ is used. The gravitational forcing is applied by the consistent scheme, see scheme (1) in Section 2.3.8.

The computation uses a 2D domain decomposition with two domains in x-direction and four domains in the periodic y-direction. It utilizes eight GPUs.

Since not the exact parameter set of the experiment is simulated, but rather non-dimensional numbers are matched, normalized results are shown. A reference velocity U_0 for this setup is reported in [184] as

$$U_0 = \sqrt{gL\frac{\Delta T}{T_0}}. \tag{6.21}$$

Turbulent statistics were collected for about $(U_0 t)/L = 185$. With the assumptions that the length of one circumvention of the cavity being $4L$ and the average velocity being $0.1U_0$ (see Fig. 6.11) the number of turnovers is about $0.1U_0 t/(4L) = 4.5$. The reference simulation reports a much shorter averaging time that results only half a turnover.

Fig. 6.10 shows field data of the simulation results. The iso-surface of the Q criterion visualizes turbulent structures. Both hot and cold walls show similar turbulent boundary layer flows. The comparison between the simulation of Salinas-Vázquez

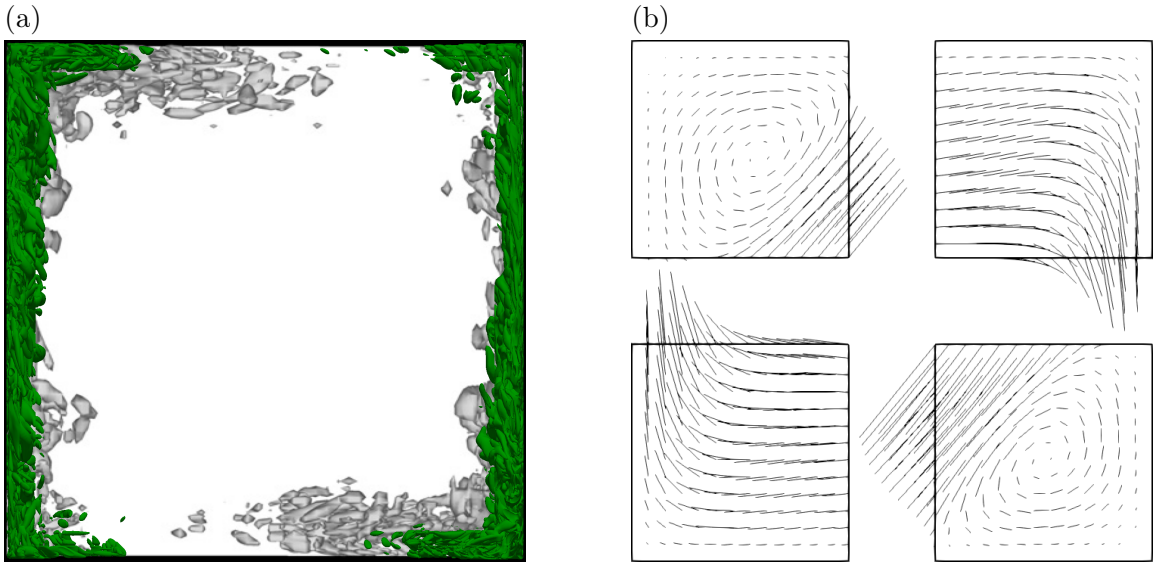


Figure 6.10: 3D Square cavity: (a) Contour of the Q criterion at $Q = 3U_0$. Grey contour: simulation by Salinas-Vázquez et al. [186]; Green Contour: present GKS simulation; The grey contour shows additional turbulence at top and bottom walls, which is not seen in the present simulation. (b) Time averaged velocity field in the four corners of the cavity. Top-left and bottom-right corners show vortices that are not visible on bottom-left and top-right. This corresponds to the experimental findings [184].

et al. [186] and the present GKS simulation reveals a difference in turbulent structures. The GKS simulation predicts a smaller region of turbulence at top and and bottom walls. This will be discussed further below, while investigating profiles of turbulent fluctuations.

The experiments investigated the time averaged flow field at the center plane of the cavity. It was found that the corners at the beginning (in flow direction) of the isothermal walls (both hot and cold) show no vortices, whereas the corners at the end of these walls show small vortices [184]. The same effect is observed in the GKS simulations, see Fig. 6.10, (b).

For the purpose of validation of the present GKS flow solver, time averaged profiles through the cavity are compared with both the reference simulation and the experimental data[1]. The vertical velocity, profiles show overall very good agreement between between simulations and experiments, see Fig. 6.11 (a). In details the simulations differ. At the end of the isothermal walls the present GKS simulations under predict the velocity, whereas the reference simulations slightly over predicts the velocity. Also at the mid height of the cavity, the present GKS captures the boundary layer profile with slightly lower accuracy.

For the horizontal velocity, agreement between experiments and simulations is poor, see Fig. 6.11 (b). At mid height, the GKS simulation accurately predicts the zero ve-

[1]Martin Salinas-Vázquez was so kind to provide both data sets.

locity, whereas the reference simulation predicts a non-zero velocity. At the beginning of the left and right walls both simulations capture the impinging flow well. The flow pattern in between hot and cold walls at both top and bottom is captured poorly by both simulations.

Temperature profiles are shown in Fig. 6.12 (a). The overall agreement between the simulations and the experiments is good. The GKS simulation captures the temperature not as good as the simulation of Salinas-Vázquez et al. [186]. Especially at the end of the left and right walls, the GKS simulation shows an under shoot in the top left corner and, respectively, an over shoot in the bottom right corner.

Finally, as a measure for turbulence, the vertical root-mean-square velocity is investigated, see Fig. 6.12 (b). The reference simulation shows an over prediction of velocity fluctuations. The present GKS simulations capture the profiles much better. Especially at beginning and end of the left and right walls, the magnitude of fluctuations is well recovered. Also the low fluctuations at top and bottom center are recovered better. The large difference in turbulent fluctuations at the end of the isothermal walls is consistent with Fig. 6.10 (a), where more turbulence is found in the simulation of Salinas-Vazquez et al. than in the present GKS simulation.

In this section turbulent natural convection in a square cavity with differentially heated walls is investigated. The comparison to experimental data shows good overall agreement and, hence, proving the validity of the present flow solver for this kind of simulations. While the simulation of Salinas-Vázquez et al. [186] agrees slightly better with he experiments for the first order statistics velocity and temperature, the second order statistic velocity fluctuations is captured with better agreement in the GKS simulation.

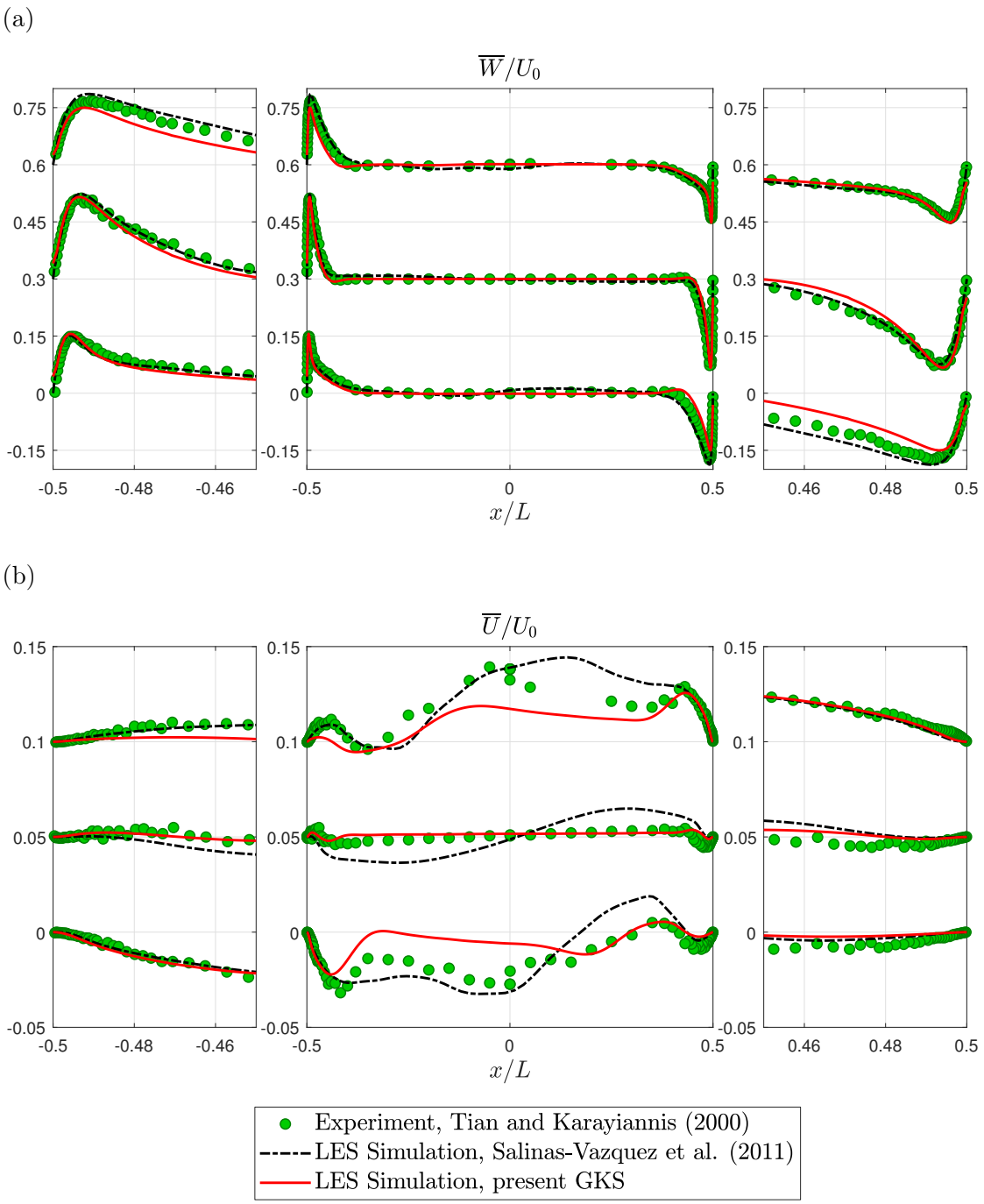


Figure 6.11: 3D Square cavity: velocity profiles at heights $z/L = 0.1$, $z/L = 0.5$ and $z/L = 0.9$ from bottom to top: (a) vertical velocity profiles. For visualization the profiles are shifted by 0.3. (b) horizontal velocity profiles. For visualization the profiles are shifted by 0.05. Experimental values from Tian and Karayiannis [184], simulation values from Salinas-Vázquez et al. [186].

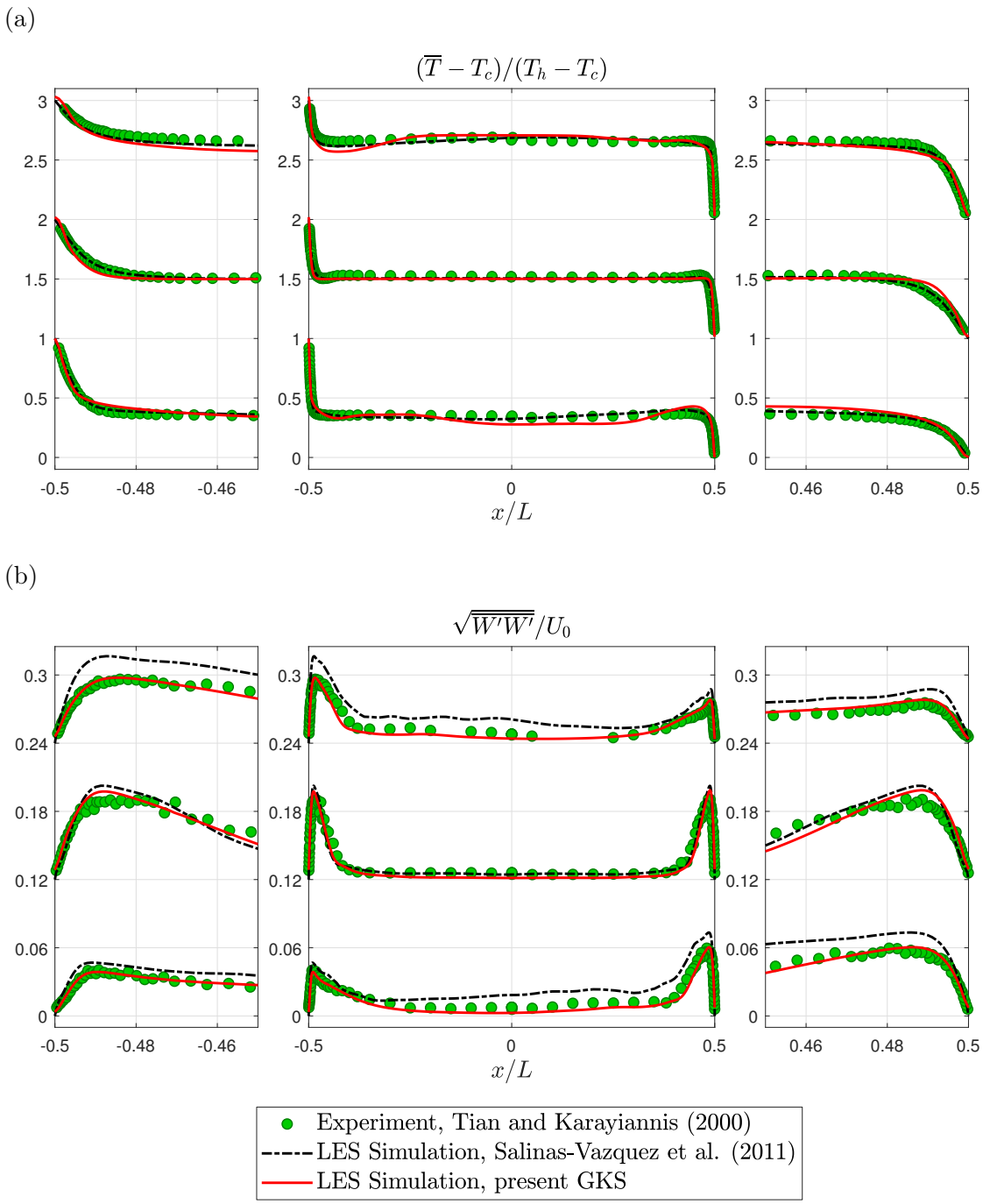


Figure 6.12: 3D Square cavity: temperature and velocity fluctuation profiles at heights $z/L = 0.1$, $z/L = 0.5$ and $z/L = 0.9$ from bottom to top: (a) temperature profiles. For visualization the profiles are shifted by 1. (b) vertical velocity fluctuation profiles. For visualization the profiles are shifted by 0.12. Experimental values from Tian and Karayiannis [184], simulation values from Salinas-Vázquez et al. [186].

6.2.3 Purdue flame

After showing the validity of VirtualFluidsGKS for turbulent natural convection, this section, and the following one, will investigate the capability of the present GKS solver for combustion driven natural convection. An extensive collection of fire benchmarks with numerical and experimental references is provided by the developers of the Fire Dynamics Simulator (FDS) in the Fire Dynamics Simulator Technical Reference Guide Volume 3: Validation [130]. The document investigates the performance of FDS against more than fifty experiments, validating the various components of the FDS software package.

For the validation of the present GKS Code, two experiments of free fire plumes are chosen. This section looks at the small scale Purdue flame, while the following section looks at the large scale Sandia flames.

The Purdue flame refers to an experiment that was performed at Purdue University, USA, specifically for validation of FDS. The experimental results were published by Xin et al. in 2005 [133] together with the validation of a (at that time) new mixture fraction model in FDS. The experiment features a methane flame over a diffuser burner with a diameter of 7.1 cm. The velocity and mass flow rate are reported as $U_{CH_4} = 0.0314\,\mathrm{m\,s^{-1}}$ and $\dot{m}_{CH_4} = 84.3 \cdot 10^{-6}\,\mathrm{kg\,s^{-1}}$, yielding the methane density as $\rho_{CH_4} = 0.68\,\mathrm{kg\,m^{-3}}$. The total heat release rate is reported to be $\dot{q} = 4.2\,\mathrm{kW}$. The specific heat of combustion can be obtained from this as $\Delta h_{CH_4} = \dot{q}/\dot{m}_{CH_4} = 50\,\mathrm{MJ\,kg^{-1}}$. In the experiment, particle image velocity is used to measure flow fields. Furthermore, species concentrations are measured.

FDS is available for download[2]. The FDS results[3] and the experimental data[4] used for validation are also available online. In order to have more control over the output data and to measure computation times, the FDS results were recomputed.

The numerical setup for FDS and GKS simulations was chosen as similar as possible. The flow domain has a size of $0.1 \times 0.1 \times 0.4\,\mathrm{m^3}$ for the FDS simulations and $0.15 \times 0.15 \times 0.4\,\mathrm{m^3}$ for the GKS simulations. The larger lateral extend was chosen to increase distance to the lateral boundaries. Both simulations use a uniform cartesian grid with grid sizes of $\Delta x = 0.004\,\mathrm{m}$ and $\Delta x = 0.002\,\mathrm{m}$ for the coarse and fine simulations, respectively. The FDS simulation applies the same open boundary condition on all walls apart from the burner. In the GKS simulation the advanced outflow boundary condition from Section 2.5.3 is used for the outflow on top of the domain, whereas the open boundary condition from Section 2.5.4 is used on the lateral sides. The bottom boundary is an adiabatic wall. The fact that flow cannot leave the domain through the lateral side, motivates the increased domain size. The burner is model by a creeping mass flux boundary condition, see Section 2.5.7 on top of a symmetry boundary condition, see Section 2.5.6.

[2] `https://pages.nist.gov/fds-smv/downloads.html`
[3] `https://github.com/firemodels/out`
[4] `https://github.com/firemodels/exp`

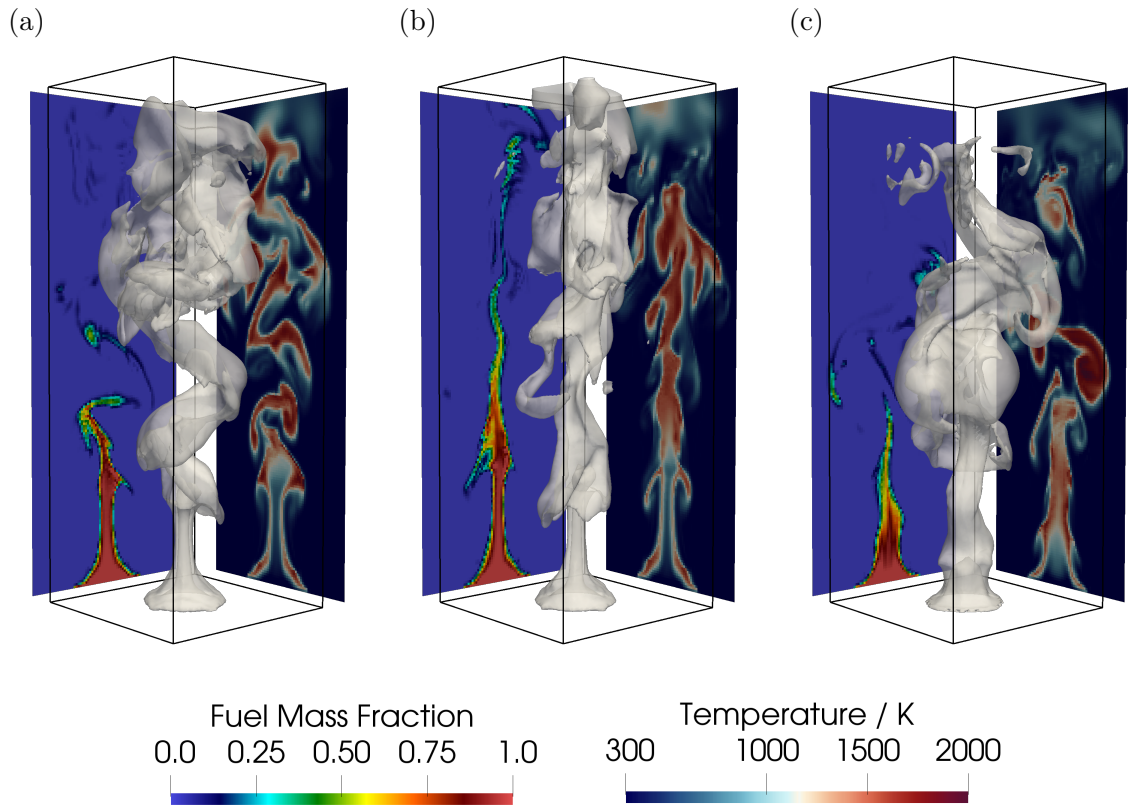


Figure 6.13: Purdue flame: instantaneous solution. The three figures show the flame at $t = 10\,\mathrm{s}$ (a), $t = 15\,\mathrm{s}$ (b) and $t = 20\,\mathrm{s}$ (c). The figures show the iso-surface of temperature at $T = 1000\,\mathrm{K}$. Further they show slices through the center of the flame, projected outwards for visualization. The left slice shows the fuel mass fraction and the right slice shows the temperature.

The parameters for the GKS simulation are air density $\rho = 1.2\,\mathrm{kg\,m^{-3}}$, gravitational acceleration $g = 9.81\,\mathrm{m\,s^{-2}}$, viscosity $\mu = 1.8 \cdot 10^{-5}\,\mathrm{kg\,m^{-1}\,s^{-1}}$, $Pr = 0.71$ and $K = 2$. For viscosity (and implicitly thermal conductivity by constant Prandtl number) Sutherland's law of temperature dependent viscosity is applied. The temperature is scaled down by a factor of 100 in order to artificially increase Mach number. Likewise the heat of combustion is scaled down. For the gravitational volume force the forcing scheme (3) based on preservation of temperature is used, see Section 2.3.8. Further, Smagorinskys eddy viscosity model with $C_S = 0.2$ is applied.

To keep the simulation stable several limiters are activated. The heat release rate limiter is set to the large value $500\,\mathrm{MW\,m^{-3}}$. The temperature limiter is set to 10^{-8} and the passive scalar limiter is activated. The effect of the temperature limiter will briefly be discussed at the end of this section.

An overview over the solution of the Purdue flame is shown in Fig. 6.13. The figure shows the temperature contour a $T = 1000\,\mathrm{K}$ at times $t = 10\,\mathrm{s}$, $15\,\mathrm{s}$ and $20\,\mathrm{s}$. Additionally, temperature and fuel mass fraction in the center of the flame are shown as

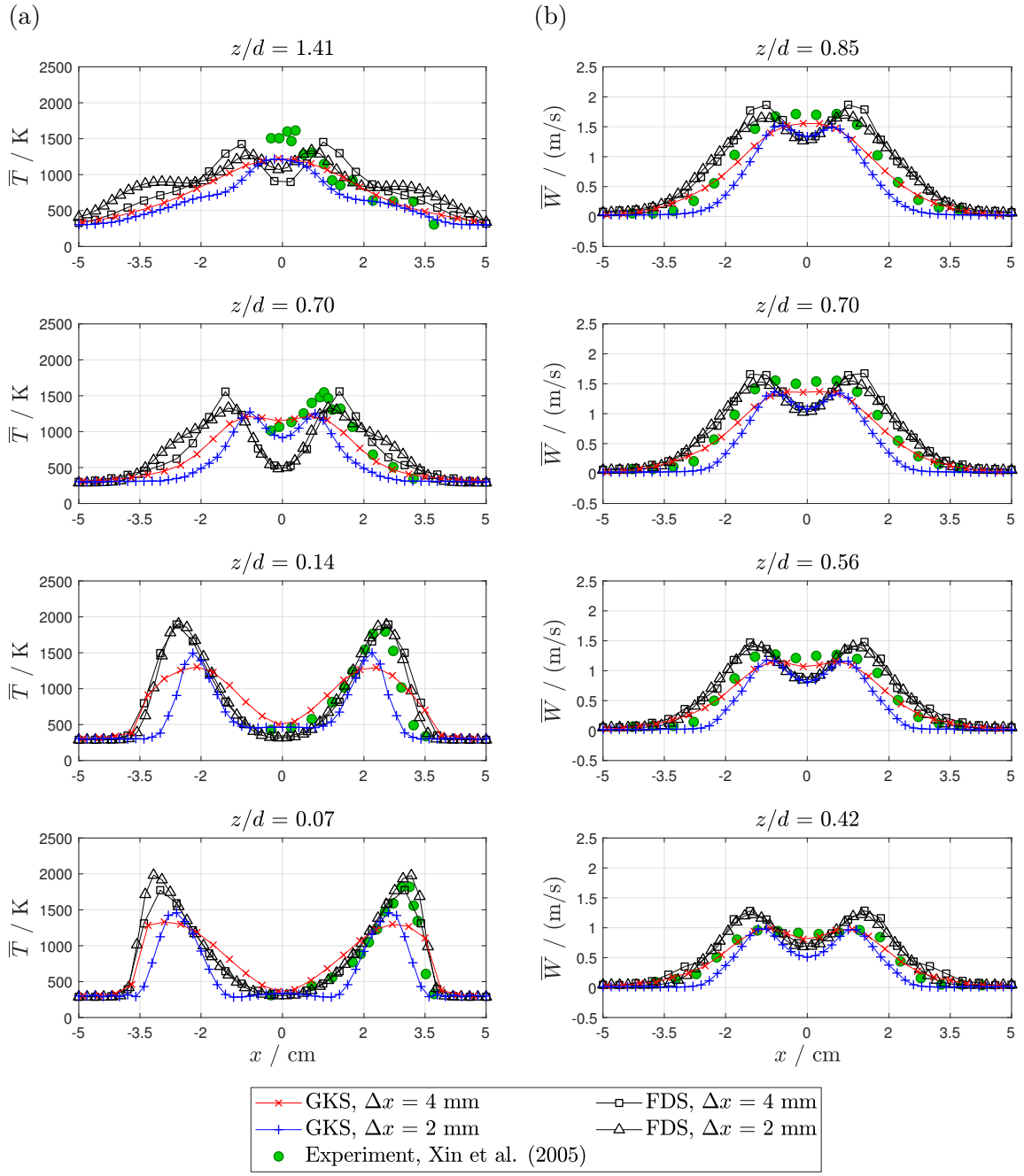


Figure 6.14: Purdue flame: temperature and vertical velocity profiles. The experimental data is available online at `https://github.com/firemodels/exp/tree/master/Purdue_Flames`.

color maps that are projected outwards for visualization.

For validation, time averaged profiles through the flame are collected starting after 10 s of flame evolution. Statistics in FDS are collected for another 10 s, while, due to shorter run times, statistics for GKS were collected for another 30 s.

A primary quantity in the simulation of fire is the flame temperature, see Fig. 6.14, (a). Experimental data for temperature is only available for half the profiles. Theoretically, the time averaged flame should be rotationally symmetric. The experimental results do not exactly show this feature though (see velocity profiles in Fig. 6.14, (b)). Two observations regarding the temperature are in place. First, in the lower profiles, the GKS simulations under predict the temperature substantially. In the higher profiles this tendency is less pronounced. FDS on the other hand captures the temperature magnitude well. Second, the flame width for the GKS simulation is smaller than for the experimental data and the FDS simulation.

The first point is an indisputable shortcoming of the present GKS simulation. The second point is disputable, though. Fig. 6.14, (b), shows the vertical velocity profiles at different heights. Note that the heights are not the same as for the temperature profiles. The velocity magnitude is predicted well by both FDS and GKS. In the original experimental data the velocity profile is not centered. Due to this, the width of the plume is captured well by FDS on the right, whereas GKS captures the shape of the velocity profile better on the left (in the lowest profile it is the other way round). In order to correct for this all profiles are shifted by 0.28 cm, such that the velocity profiles are centered. The shift is computed based on the normalized first moment of the velocity distribution. After shifting, the experimental velocity profile lies in between the FDS and GKS results. The shift does not correct the flame width deviation of the GKS. The asymmetry reduces the certainty of the experimental data. Under the given uncertainty velocity and temperature profiles show acceptable agreement to the experimental data.

A further observation is that FDS results of 0.004 m and 0.002 m nearly collapse, while the GKS results show a larger discrepancy between the resolutions.

Horizontal velocities are also investigated, see Fig. 6.15, (a). The two GKS simulations and the fine FDS simulation agree well to each other, but as a group slightly under estimate the peak of horizontal velocity. The fact that the coarse FDS simulation predicts the velocity magnitude better cannot be trusted, since this effect disappears under refinement. No quality distinction between FDS and GKS can be concluded from the horizontal velocity profiles.

Finally, the three root-mean-square velocity fluctuation profiles are shown in Fig. 6.15, (b). The disagreement between experimental data and simulations is substantial. It is noteworthy that the coarse GKS simulation fits the profiles better than the fine one.

Concluding, the accuracy of the present GKS can at least compete with FDS. The uncertainty of the experimental data leaves much space for speculation, but this test case shows that the present GKS obtains in principle correct flow features.

For the Purdue flame the temperature limiter was used. Not using the temperature limiter results in unstable simulations. The effect of the temperature limiter is visualized in Fig. 6.16. The two figures show the plume right after the start of the simulation without limiter (a) and with limiter (b). At the leading front of the plume energy is accumulated, leading to a large increase in temperature. The limiter

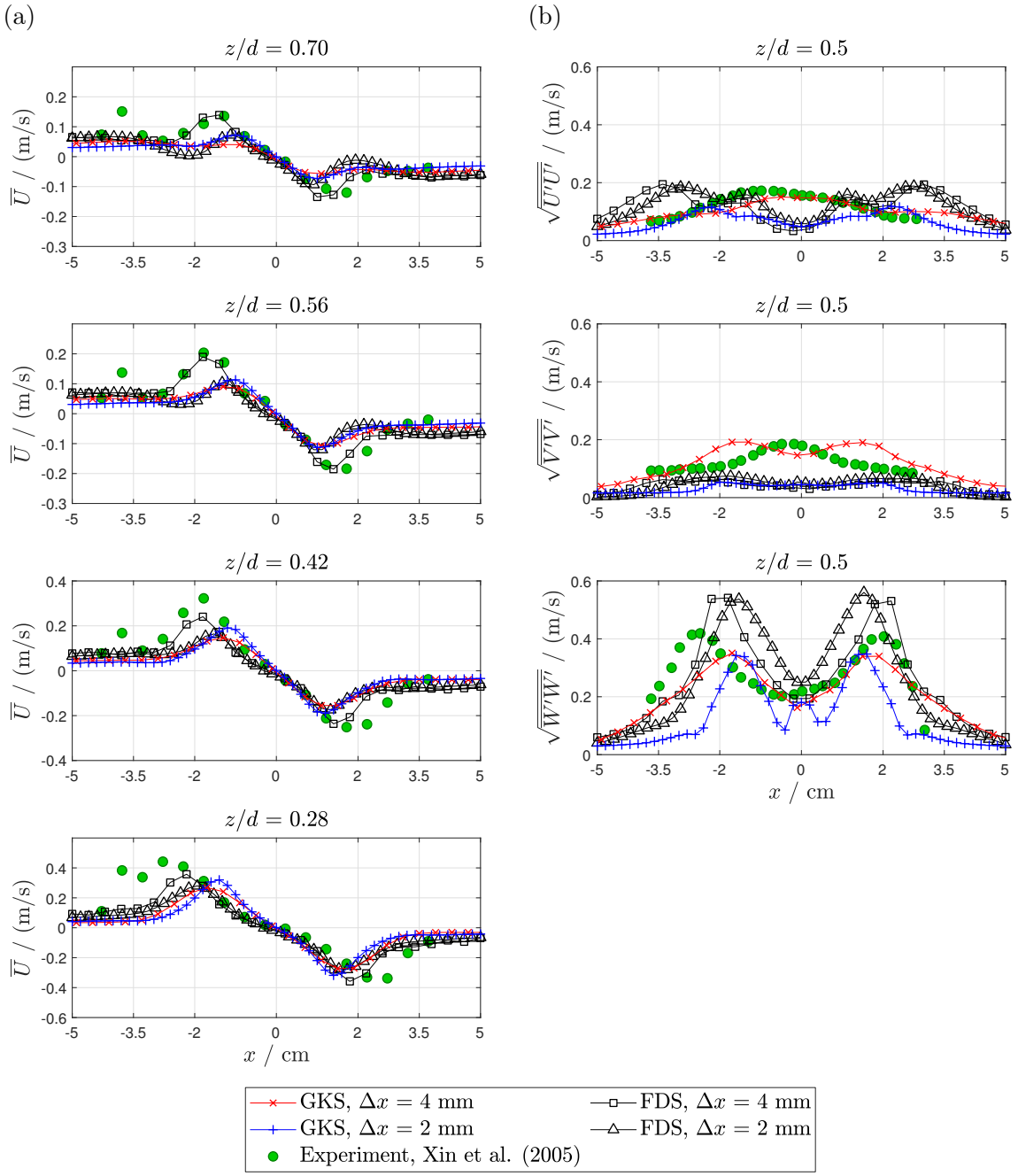


Figure 6.15: Purdue flame: horizontal velocity and fluctuation profiles. The experimental data is available online at `https://github.com/firemodels/exp/tree/master/Purdue_Flames`.

increases heat diffusion in the presence of large temperature gradients, maintaining energy conservation, but preventing harmful gradients.

In order to conclude the section on the Purdue flame, run times are investigated. The GKS simulations were performed on single NVIDIA Tesla P100 GPUs. The FDS sim-

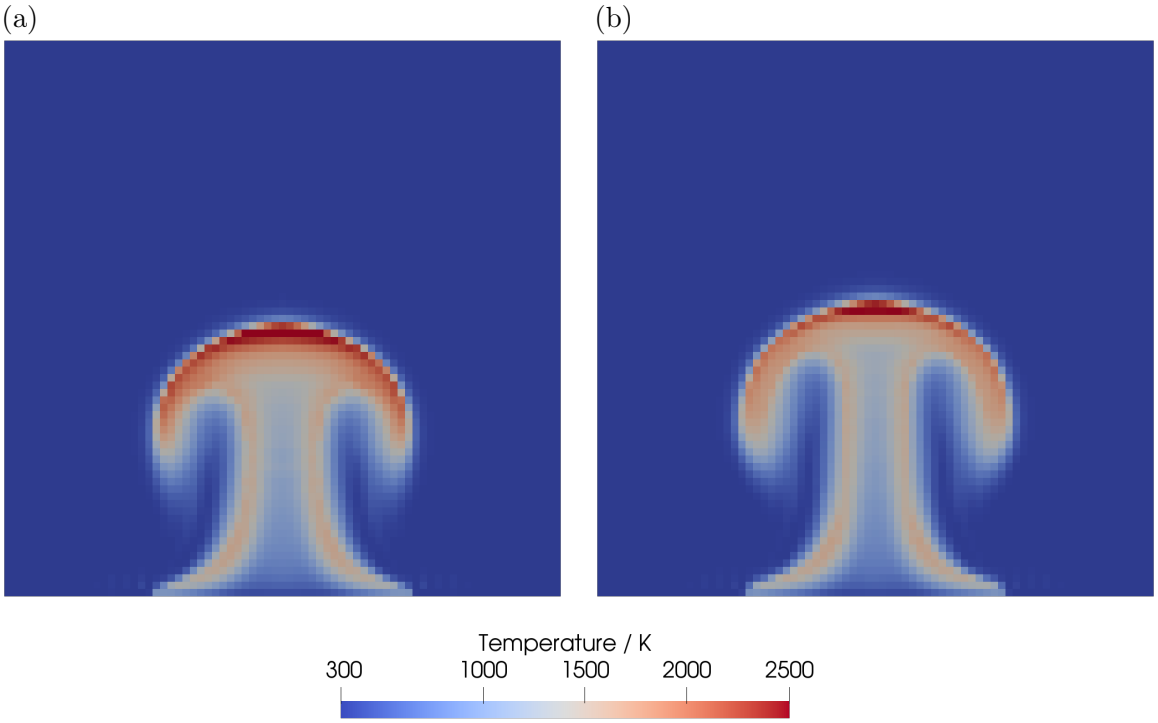


Figure 6.16: Purdue flame: left without temperature limiter and right with temperature limiter; $t = 0.15\,\mathrm{s}$

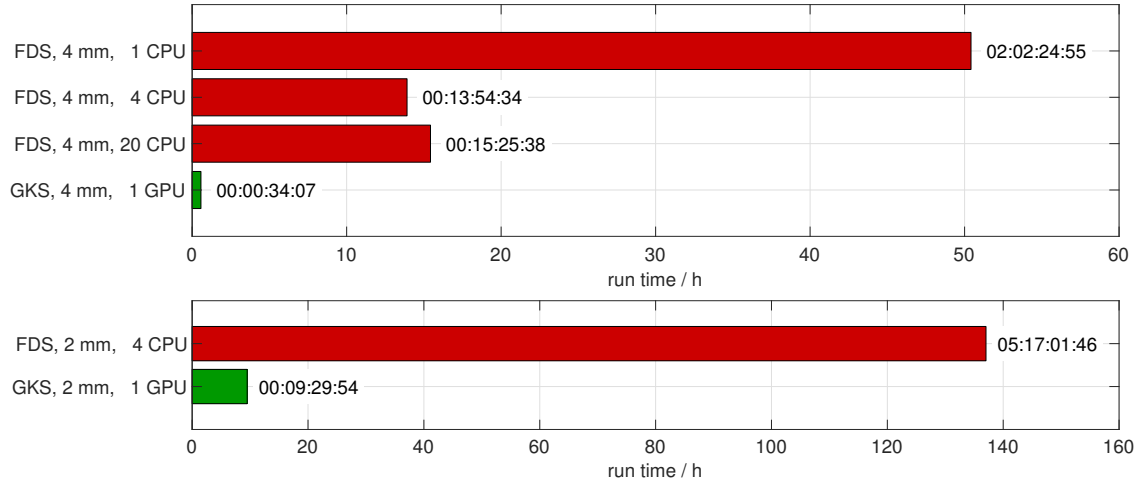


Figure 6.17: Purdue flame: time required for FDS and GKS simulations to simulate flame evolution for 20 s

ulations were performed on single CPUs with multiple cores utilizing multi threading via OpenMP. For the sake of fairness it has to be emphasized that MPI parallelism for FDS was not investigated, even though being supported and being reported to yield better speedups. Run times were measured as wall clock time required to simulate flame evolution over 20 s. Run times for both resolutions and both methods are given

in Fig. 6.17. The first glance observation is that VirtualFluidsGKS is faster by more than an order of magnitude. The coarse FDS simulation was computed with a single core, four and twenty cores. While the step from a single to four cores brings nearly ideal parallel efficiency, a further increase in corse brings no benefit for this simulation. Larger systems might benefit more, though. The speedup from FDS to GPU-GKS is about 24 for the coarse resolution and about 14 for the fine resolution. The two GKS results show an increase of factor 16 in computation time, which is to be expected due to eight times the number of cells and twice the number of time steps. FDS scales somewhat better in that regard, because the fine simulation requires only ten times the time of the coarse. One reason for this could be the use of adaptive time stepping based on actual CFL number used by FDS, where GKS uses a constant CFL number.

This test case shows that the present GKS can be used for fast fire simulations. The accuracy is not as good as in the case of FDS, but not much worse either. The following section investigates fire plumes on larger scale.

6.2.4 Sandia flames

Detailed data sets of large scale fire plumes were collected at Sandia National Laboratories, USA, by Tieszen et al. [187, 188]. While the Purdue flame in the prior section is basically a table top experiment, the Sandia Flames require large facilities. Similar to the Purdue flame, these experiments were designed specifically for the validation of CFD codes.

The experimental set up features a circular burner with a diameter of 1 m. The fuel is again methane. As an additional fuel hydrogen was investigated, but is not considered in this work. The experiment was repeated with several heat release rates, i.e. several fuel mass fluxes, see [188] for an overview. Particle image velocimetry is used to measure velocities. Data is collected for 7 s.

Again, FDS and GKS computations for these tests are compared. Both utilize uniform cartesian grids with cell sizes of $\Delta x = 6.25\,\mathrm{cm}$, $\Delta x = 3.125\,\mathrm{cm}$ and $\Delta x = 1.5625\,\mathrm{cm}$. The domain has a size of $3 \times 3 \times 4\,\mathrm{m}^3$. The boundary conditions are similar to the Purdue flame simulation. The parameters for the GKS simulation are (again) air density $\rho = 1.2\,\mathrm{kg\,m^{-3}}$, gravitational acceleration $g = 9.81\,\mathrm{m\,s^{-2}}$, viscosity $\mu = 1.8 \cdot 10^{-5}\,\mathrm{kg\,m^{-1}\,s^{-1}}$, $Pr = 0.71$ and $K = 2$. The heat release rate limiter is set to $2\,\mathrm{MW\,m^{-3}}$. The temperature limiter is set to 10^{-3}.

Table 6.4: Parameters for the Sandia flame test cases

Test	description	$\rho_{CH_4}/\,\mathrm{kg\,m^{-3}}$	$\dot{m}_{CH_4}/\,\mathrm{kg\,s^{-1}}$	$\dot{q}/\,\mathrm{MW}$
Test 14	low flow rate	0.5405	0.0314	1.57
Test 24	medium flow rate	0.5464	0.0416	2.08
Test 17	high flow rate	0.5641	0.0518	2.59

Of the several tests performed during the experimental study, the Tests 14, 24 and 17, which are characterized by low, medium and high fuel flow rates, respectively, are chosen for the validation. The fuel flow rates and resulting heat release rates are listed in Table 6.4. Due to the amount of result data, only the results for Test 24 are shown here in this section. Results for the Tests 14 and 17 are shown in Appendix D.

The temperature fields of both simulation approaches are shown in Fig. 6.18. The FDS solution (a) shows a lot of detail in the temperature field, including steep temperature gradients. The GKS solution (b) on the contrary shows a smooth temperature field, which reaches the same magnitude in temperature, but no fine details are visible.

For the validation turbulent statistics are collected again between $t = 10\,\mathrm{s}$ and $20\,\mathrm{s}$ for FDS and between $t = 10\,\mathrm{s}$ and $40\,\mathrm{s}$ for GKS. Experimental data is only available for velocity and turbulent kinetic energy. Nevertheless, temperature profiles are shown in figure Fig. 6.19 (a). Similar to the Purdue flame, the FDS results of different resolutions nearly collapse. The GKS simulations show more variation between the resolutions. The temperature magnitude of the fine GKS simulation agrees well with the FDS result, whereas the coarser simulations over predict temperature. Regarding the shape of the profile, higher resolution for GKS yields a narrower flame. Also

(a) (b)

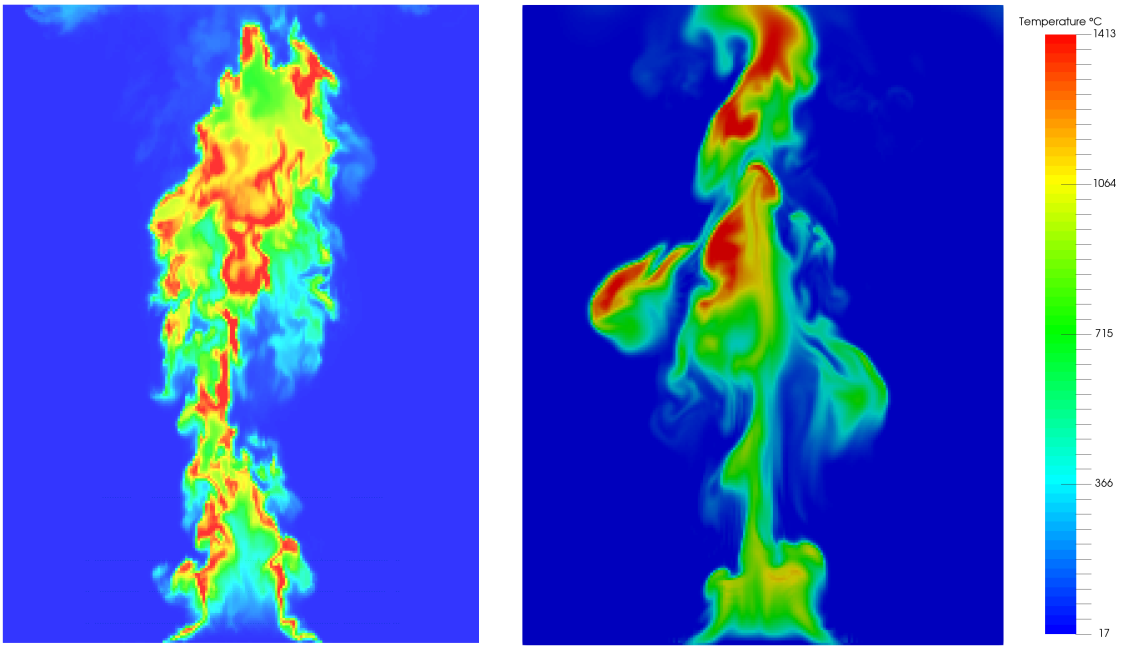


Figure 6.18: Sandia flame, Test 24: Instantaneous temperature field computed by FDS (a) and VirtualFluidsGKS (b) at $t = 10\,\mathrm{s}$ from the fine simulations with $\Delta x = 1.5625\,\mathrm{cm}$.

the temperature drop in the center of the flame is much more pronounced in the FDS simulation, than for GKS. The absence of experimental data, makes a conclusion from this comparison difficult.

The vertical velocity profiles are shown in Fig. 6.19 (b). Again, the GKS profiles show a greater discrepancy, than the FDS results. Right above the burner, both FDS and GKS agree well on velocity magnitude. GKS predicts a lower velocity in the center of the flame, though. At the second height, the fine GKS simulation collapses with the experimental data both in shape, as well as in magnitude. Only in the middle it predicts a slightly stronger temperature drop. At greater heights, the velocity magnitude is still predicted with good accuracy, but the GKS profiles are narrower than measured in the experiment. Overall, the VirtualFluidsGKS and FDS show similar agreement to the measured velocities.

The time averaged horizontal velocities do not give a lot of insight, see Fig. 6.20 (a). With refinement, the GKS profiles tend towards the measurements. A more relevant quantity is the turbulent kinetic energy $\mathrm{TKE} = (\overline{U'U'} + \overline{V'V'} + \overline{W'W'})/2$, especially with Fig. 6.18 in mind. The temperature fields suggest that the FDS solution is much more turbulent than the GKS solution. The profiles in Fig. 6.20 (b) confirm this statement only partially. Directly above the burner, GKS shows no turbulence, while FDS does. The coarsest GKS resolution continues to show near to no turbulent kinetic energy also a larger heights. The medium and fine resolutions lack TKE towards the

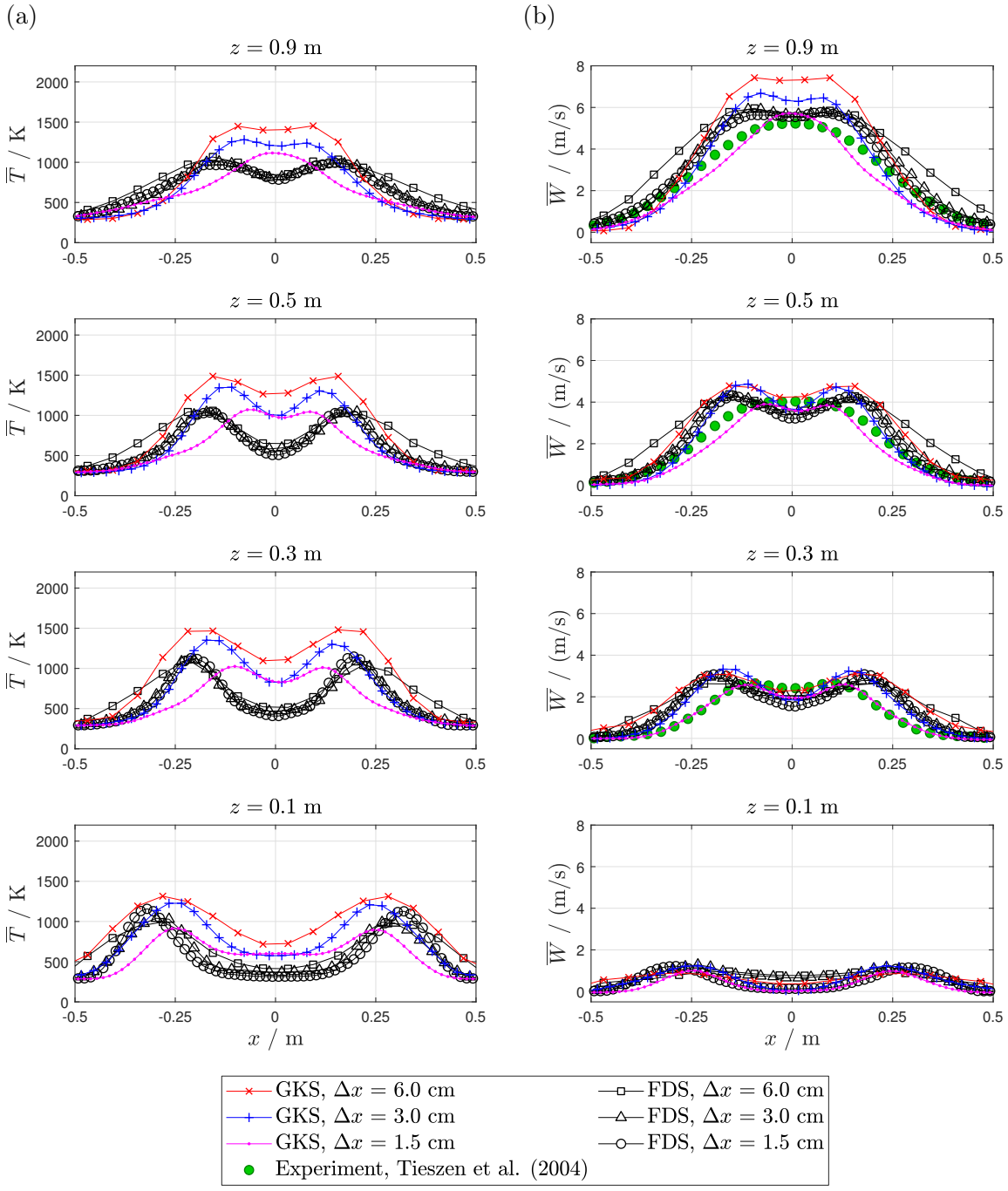


Figure 6.19: Sandia flame, Test 24: temperature and vertical velocity profiles

middle of the flame, but fit well to the shape of the experimental profiles. The over all agreement of the fine GKS simulation with the experimental data is acceptable.

A further investigation of the turbulence in the flame can be done by investigating the turbulent energy density spectrum. For that time series of vertical velocity are collected at four points with coordinates $(x, z) = (0, 0.5)$, $(0, 2.0)$, $(0.5, 0.5)$ and $(0.5, 2.0)$. The energy density is obtained as the square of the absolute value of the

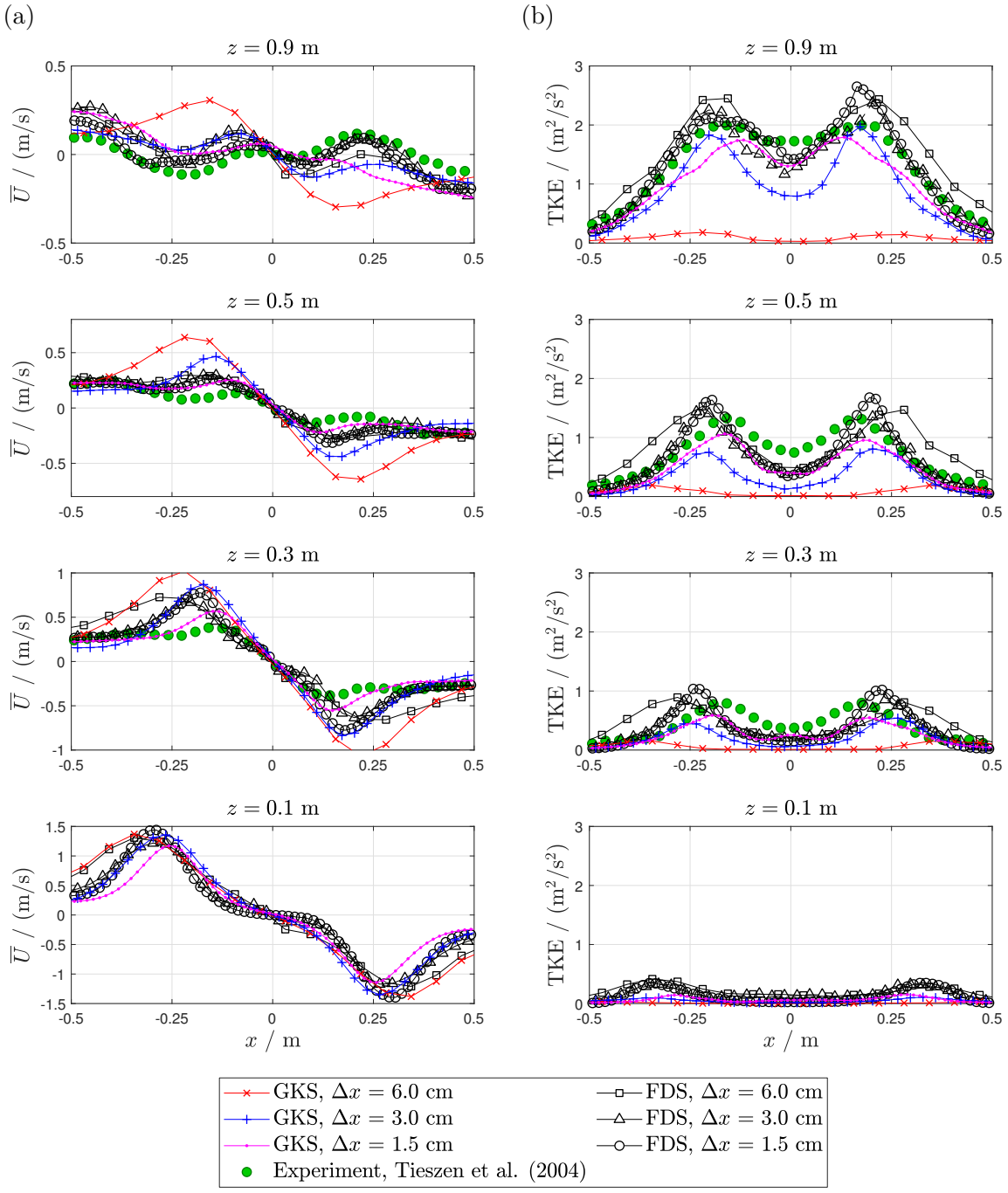


Figure 6.20: Sandia flame, Test 24: horizontal velocity and turbulent kinetic energy profiles

discrete fourier series of the vertical velocity signal.

Results for the energy spectra are shown in Fig. 6.21. The FDS spectra show the expected energy cascade with exponent $-5/3$. The GKS spectra show slightly less energy, which is consistent with the findings in Fig. 6.20 (b) that the GKS simulations resolves a little less energy. The slope of the energy cascade does not fit the exponent

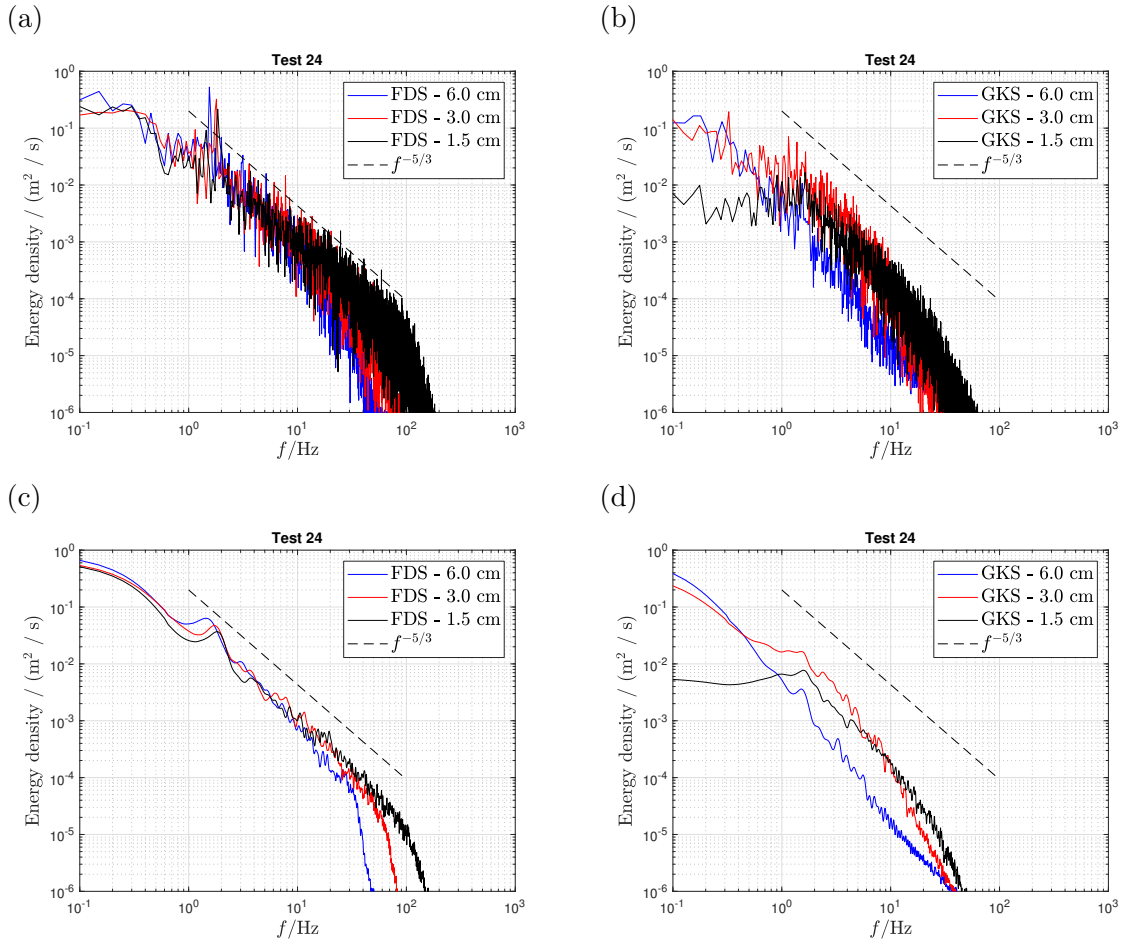


Figure 6.21: Sandia flame, Test 24: energy spectra of FDS and VirtualFluidsGKS. (a) and (b) show the original spectra and (c) and (d) the lowpass filtered spectra.

$-5/3$ perfectly. For higher frequencies the energy drops faster than expected. This suggests that the FDS flow solver is some what better suited for the simulation of turbulence, but the results of the GKS are not much worse either.

Finally, run times are compared for this test case. VirtualFluidsGKS is about 55 times fast for the coarsest resolution and about 40 times faster for the finest resolution.

This section shows that the present GKS can be used for fast fire simulations. In terms of accuracy there is room for improvement, especially with respect to the representation of turbulence.

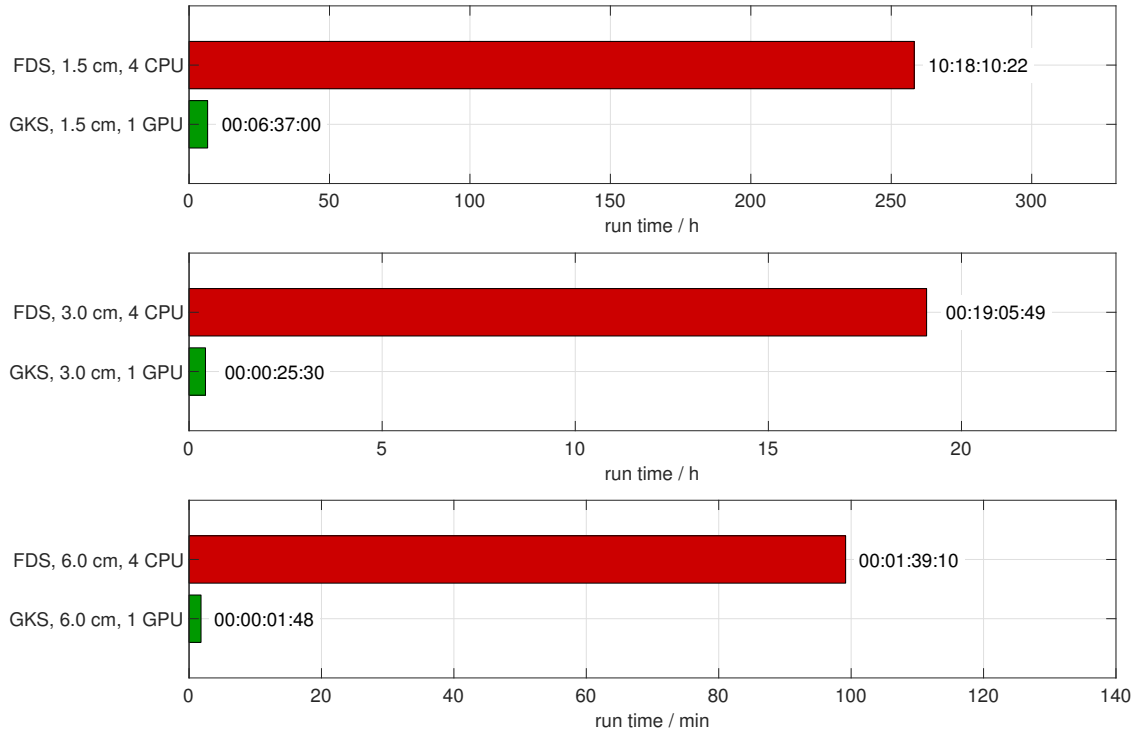


Figure 6.22: Sandia flame, Test 24: run times

6.3 Application

After having shown the validity of VirtualFluidsGKS for multiple test cases including validation of fire, this section will present two applications of VirtualFluidsGKS. The first application focuses on turbulent natural convection at large temperature differences and the second application investigates a compartment fire.

6.3.1 Boundary layer stability in cavity with differentially heated walls

In our first publication on GKS for natural convection we investigated the two-dimensional flow in a square cavity at large Rayleigh numbers and large temperature differences [15]. Lets consider a non-dimensional temperature difference of $\epsilon = (T_h - T_c)/T_0 = 1.2$. At Rayleigh numbers up to about $Ra = 10^7$ the two-dimensional flow field is stationary [169, 15]. Due to expansion of heated fluid and contraction of cooled fluid the flow field is not symmetric. At Rayleigh number $Ra = 10^8$ the flow becomes unsteady in the form that the convection jets at the boundaries impinge on top and bottom walls, such that the reflection is unsteady and oscillatory [189]. At Rayleigh number $Ra = 10^9$ the boundary jet on the hot wall developes two-dimensional turbulence, while the jet on the cold wall only shows unsteady behavior during the impingement on the insulated wall [189]. This phenomenological difference was further investigated in [15] at Rayleigh number $Ra = 5 \cdot 10^9$. The instantaneous temperature field and the time average turbulent kinetic energy of the test in [15] are shown in Fig. 6.23.

The boundary layer stability of heated and cooled boundary layers was investigated theoretically and experimentally in the past. An analysis of heated and cooled boundary layer stability is given by Schlichting and Gersten [190, Chapter 15.2.4d]. Therein, the boundary layer stability of an incompressible fluid with temperature dependent viscosity is analyzed based on the boundary layer equations. For gases the viscosity increases with temperature, while liquids show the opposite behavior. Under the assumption that the temperature decreases going away from the wall, the viscosity also decreases for a gas. This enables an inflection point in the velocity profile. On the cooled wall viscosity increases away from the wall, such that the curvature of velocity profile is negative and, hence, the velocity profile is monotonic. This leads to the conclusion that the the stability for the heated boundary layer is decreased for gases. For liquids heating would increase boundary layer stability. This argument is based solely on the temperature dependence of viscosity. These findings were confirmed experimentally by Liepmann and Fila [191] for gases and Lauchle and Gurney [192] for liquids. Lees [193] extends the above argument based on detailed theoretical analysis for compressible gases.

At a first glance this could be the explanation to the qualitatively different behavior of hot and cold wall. In order to verify this, we presented simulations with both temperature dependent viscosity (based on Sutherland's law) and simulations with

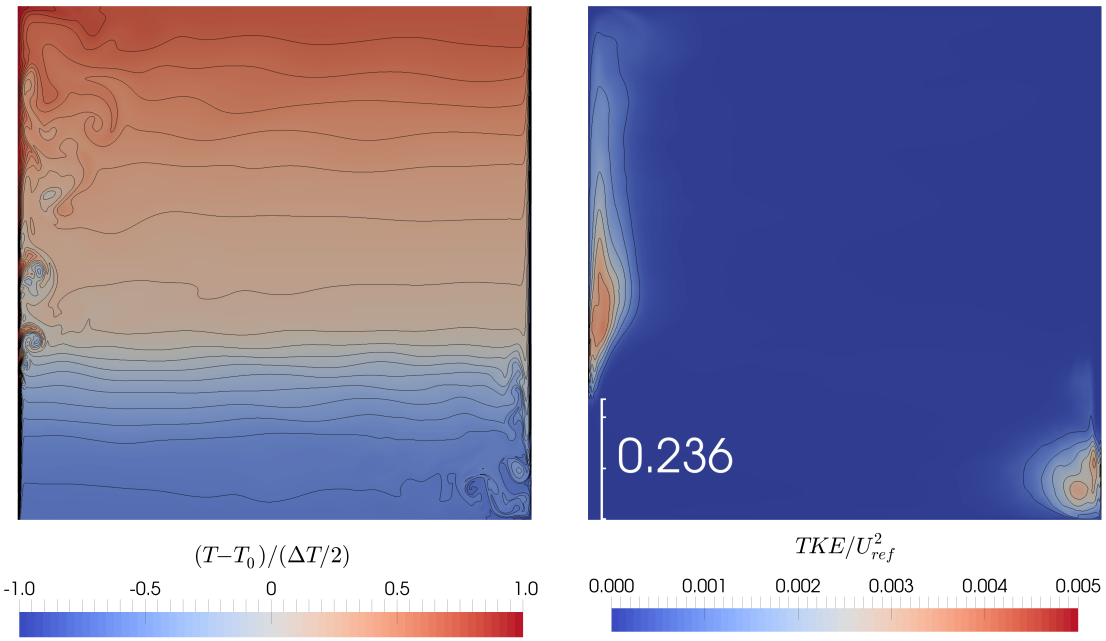


Figure 6.23: Boundary layer stability in square cavity with differentially heated walls in 2D at large temperature difference at $Ra = 5 \cdot 10^9$: The heated wall shows two dimensional turbulence, while the cooled wall only shows unsteady behavior in the region, where the boundary jet impinges on the bottom wall. Figure reproduced from [15].

constant viscosity [15]. The qualitative effect is found in both simulations. The onset of turbulence further downstream for temperature dependent viscosity, therefore somehow in line with the above argument, but it does not explain the general transition to turbulence on the hot side, which is absent on the cold side. In Lenz et al. [15] we proposed a stabilization and destabilization mechanism based on diverging flow in the hot jet due to thermal expansion and converging flow in the cold wall due to thermal contraction, respectively.

Up to this point the findings of Lenz et al. [15] were recited. It must be stressed though that investigations therein were two-dimensional, while turbulence behaves differently in three dimensions. Further, the cited theoretical and experimental investigations [190, 191, 192, 193] were done for free stream boundary layers. The dynamics in the cavity with differentially heated walls at large temperature differences are much more complex than a free stream boundary layer. First, in the cavity flow the boundary layer is characterized by a boundary jet, driven by natural convection, such that the velocity profiles will be very different from a free stream boundary layer. Second, the driving force of the jet depends only on the local temperature difference. Finally, the impingement of the boundary jet on the insulated walls and its reflection has to be taken into account. Hence, we provide new three-dimensional simulation results of a comparable setup and look deeper into the present dynamics.

The setup features a cavity with an aspect ratio H/L of two, such that the heated

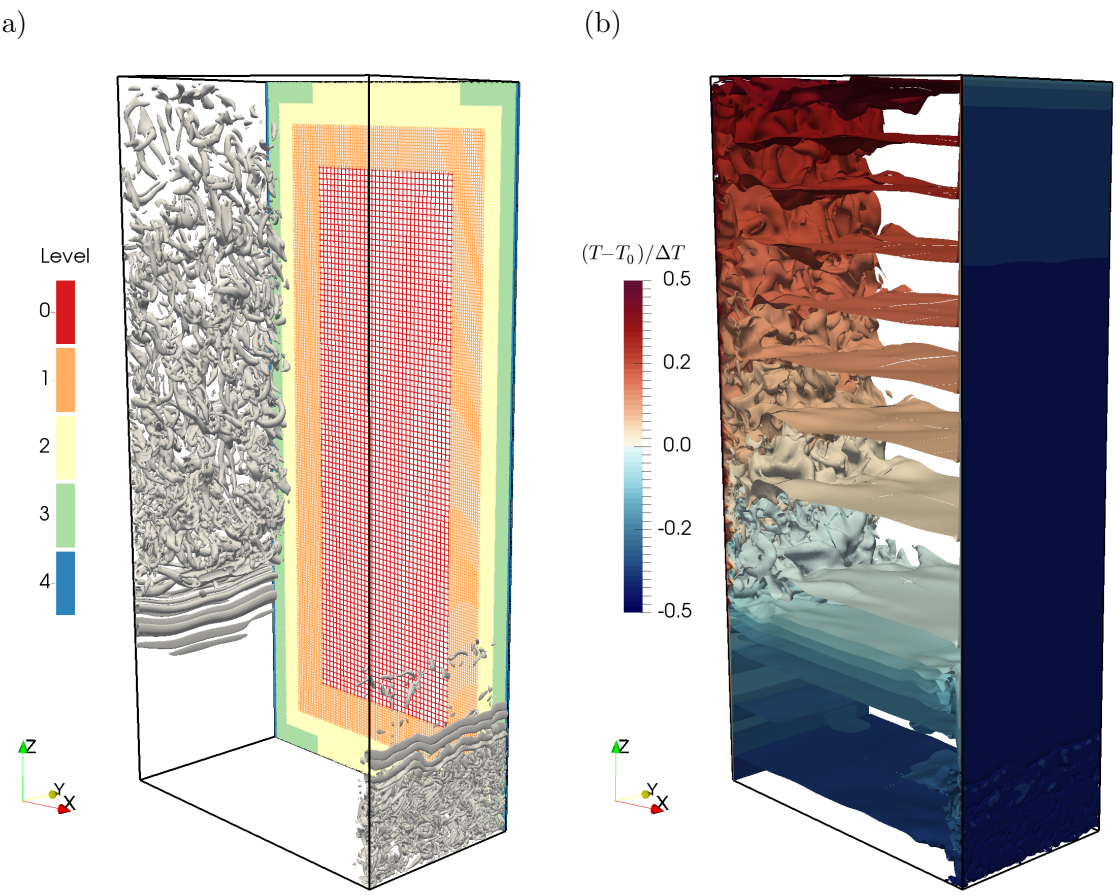


Figure 6.24: Results of boundary layer stability simulation: (a) grid and Q-criterion iso contour, (b) instantaneous temperature contours.

and cooled walls are twice as long as the insulated top and bottom walls, in order to increase the length of the convective boundary layer flow. In the third direction (here the y-direction) the domain extends for $L/2$ and the domain is periodic in this this direction. The Rayleigh number based on the distance of hot and cold walls L is $5 \cdot 10^9$. The Barometric number based on the height $H = 2L$ is $Ba = 0.2$. Material parameters are $Pr = 0.71$ and $K = 2$. The non-dimensional temperature difference is large, i.e. $\epsilon = 1.2$. For the volume force in negative z-direction the consistent scheme, see scheme (1) in Section 2.3.8, is used.

The domain was decomposed in four parts and run on four GPUs with communication along x- and y-directions. The grid has a background resolution on level 0 of $L/64$ and has four levels of refinement, such that the resolution at hot and cold walls on level 4 is $L/1024$. The first two refinement levels comprise all walls, the third level hot and cold walls plus a short part in the impingement region and the finest level comprises only the hot and cold walls. The grid with refinement levels is shown in Fig. 6.24 (a). It comprises a total of about 40 million cells. The simulation was run for 1.2 million time steps on the coarse level. In terms of free fall times, this corresponds to $1944t^*$, based on Eq. (6.12). Turbulent statistics were collected starting at time step $500,000$.

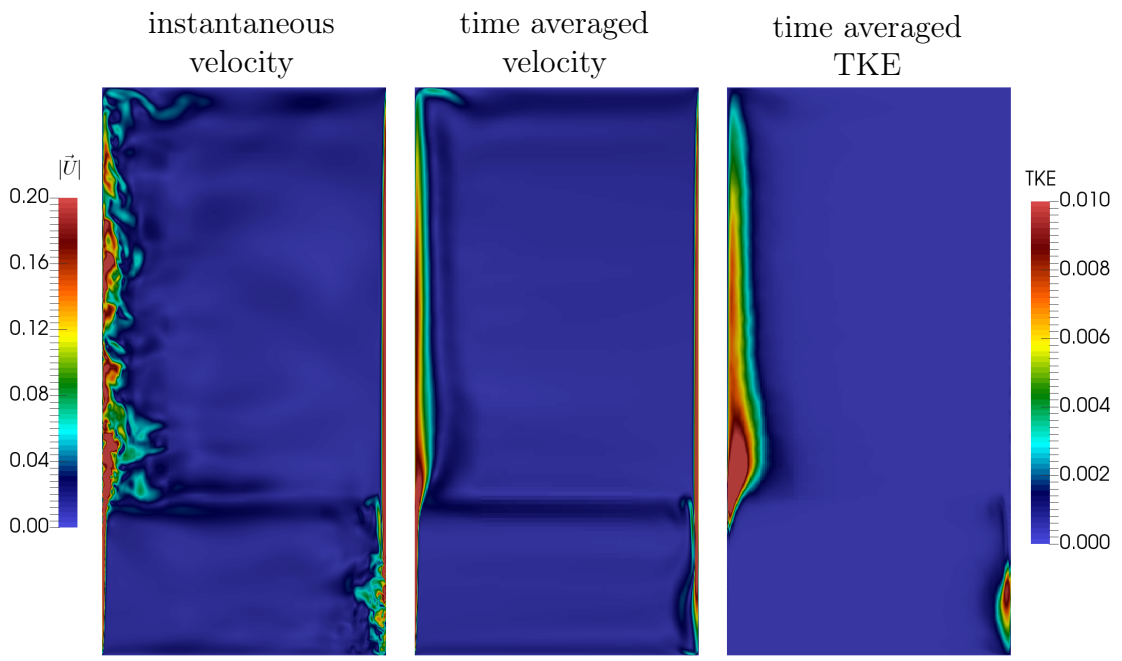


Figure 6.25: Results of boundary layer stability simulation: From left to right instantaneous velocity, time averaged velocity and time averaged turbulent kinetic energy, TKE.

The combined performance of this simulation was $1.1 \cdot 10^9$ CUPS and the wall clock time was six days and eight hours.

An overview over the instantaneous solution is given in Fig. 6.24. Sub-figure (a) visualizes the turbulence in terms of the iso-contour of the Q-criterion. The qualitative difference between hot and cold walls is clearly visible. The turbulence is present above a height of about one fifth of the height, while on the cold side turbulence only appears below about one fifth of height. Sub-figure (b) shows the temperature distribution in terms of temperature iso-surfaces. The turbulence at the hot wall is clearly visible. In the remainder of the domain the fluid is stratified with warm temperature above and cold temperature below. Two initial observations shall be made at this point. First, the height where the turbulence begins on hot and cold wall is equal. Second, at this height the steepest vertical gradient in temperature is found. The time evolution of the Q-criterion is available on YouTube under

`https://youtu.be/PsTSkPFASOI`.

Fig. 6.25 shows slice data extracted from the three-dimensional simulation. The instantaneous velocity shows the turbulence on both walls. In the time averaged velocity field the reflection of the jet in the top-left corner is clearly visible, while it is much smaller in the bottom right. The time averaged TKE clearly identifies the regions of turbulence. At the height of the beginning of the turbulence at the hot wall a

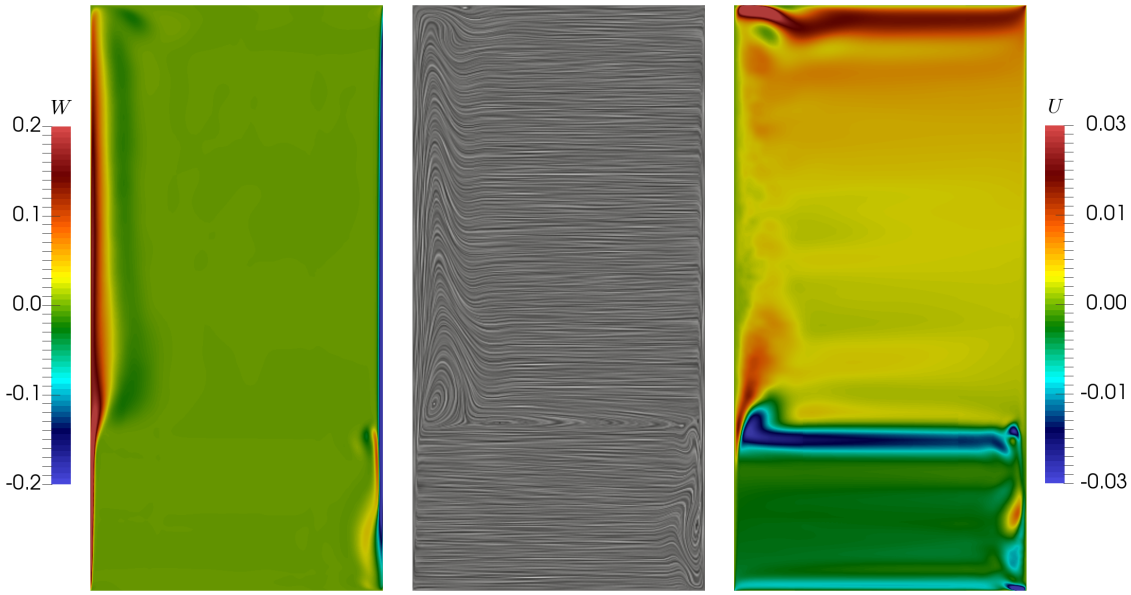


Figure 6.26: Results of boundary layer stability simulation: Velocity components on the left and right. A comparably strong negative horizontal velocity is found at the height where the flow on the hot wall becomes unstable. The figure in the middle shows a Surface LIC flow visualization of the time averaged flow field [194].

horizontal non-zero velocity is observed over the whole length of the cavity. For further investigation, the time averaged velocity components are shown separately in Fig. 6.26. The figure also shows a Surface Line Integral Convolution (Surface LIC) flow visualization of he time averaged flow field that was generated by the Surface LIC plugin in Paraview [194].

The vertical velocity W behaves mostly as expected. A strong jet is observed along the hot wall. At one point the jet widens substantially. A slight downward backflow is observed that results from the rotation of the turbulent vortices. The jet on the cold side does not show a substantial widening, but rather features a strong upward backflow in the lower region on the domain. This backflow stops at the same height, where the turbulence on the hot side starts. The horizontal velocity gives interesting insight in the dynamics of this flow problem. The hot fluid is transported from left to right mostly at the very top of the cavity. Contrary to that a strong negative velocity is observed far away from top and bottom walls. The height of this horizontal jet, coincides with the height where the turbulence on the hot wall is initiated. This suggests a new interpretation of the initiation of the turbulence. This horizontal jet impinges on the jet on the hot wall, disturbing it and thereby initiating the turbulence.

Fig. 6.27 shows the instantaneous density and temperature fields. The convective jets on both sides start in regions where the surrounding has a high and low density for hot and cold jets, respectively. Hence, the heating and cooling at the walls leads to

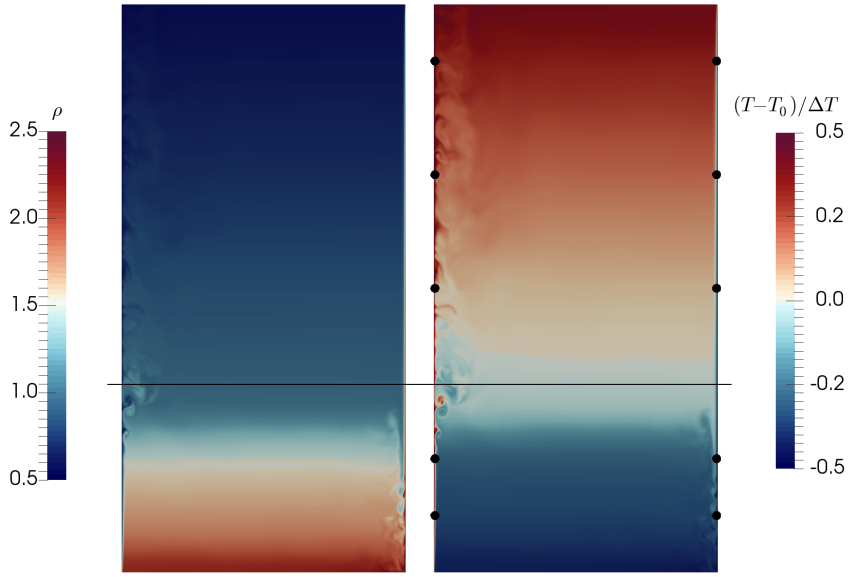


Figure 6.27: Results of boundary layer stability simulation: Instantaneous density and temperature fields. The black line vertically divides the mass in two halves. The black points denote locations where time series were recorded.

densities very different from the surroundings, such that strong forces accelerate the fluid and thereby form the jets. When entering a region with more similar densities, these forces reduce. Therefore, we find less accelerated jets in these regions. Further, the jets drag hot and cold fluid with them. We observe that the jets become turbulent when the jets enter the regions with less acceleration. The horizontal black line in the figure vertically divides the mass in two halves. This division is, however, not correlated with the onset of turbulence, which appears at lower height.

In addition to the field data, time series of vertical velocity W were recorded at several points close to the hot and cold walls. The heights of the points are $1/10H$, $2/10H$, $5/10H$, $7/10H$ and $9/10H$. The wall distance is about $0.004L$ such that the points are located in the center of the convective jets. The locations of the measure points are shown in Fig. 6.27 as black dots. The time series are shown in Fig. 6.28 (a) and (b). The lowest point on the hot wall is located in the region of laminar flow and the second point in the region of turbulent onset. Hence, the velocity at these points only varies slightly. The upper three points are in the turbulent region, which is clearly visible in the large oscillations. On the cold wall the upper three points are in the laminar region. The fourth point is in the onset region and shows slight oscillations. Only the lowest point is in the region of turbulence.

Moreover the energy spectra were computed from the time series in an interval of 1.1 million time steps. The velocity spectra are low-pass filtered to get more distinct spectra before computing the energy spectra. The spectra are shown in Fig. 6.28 (c)

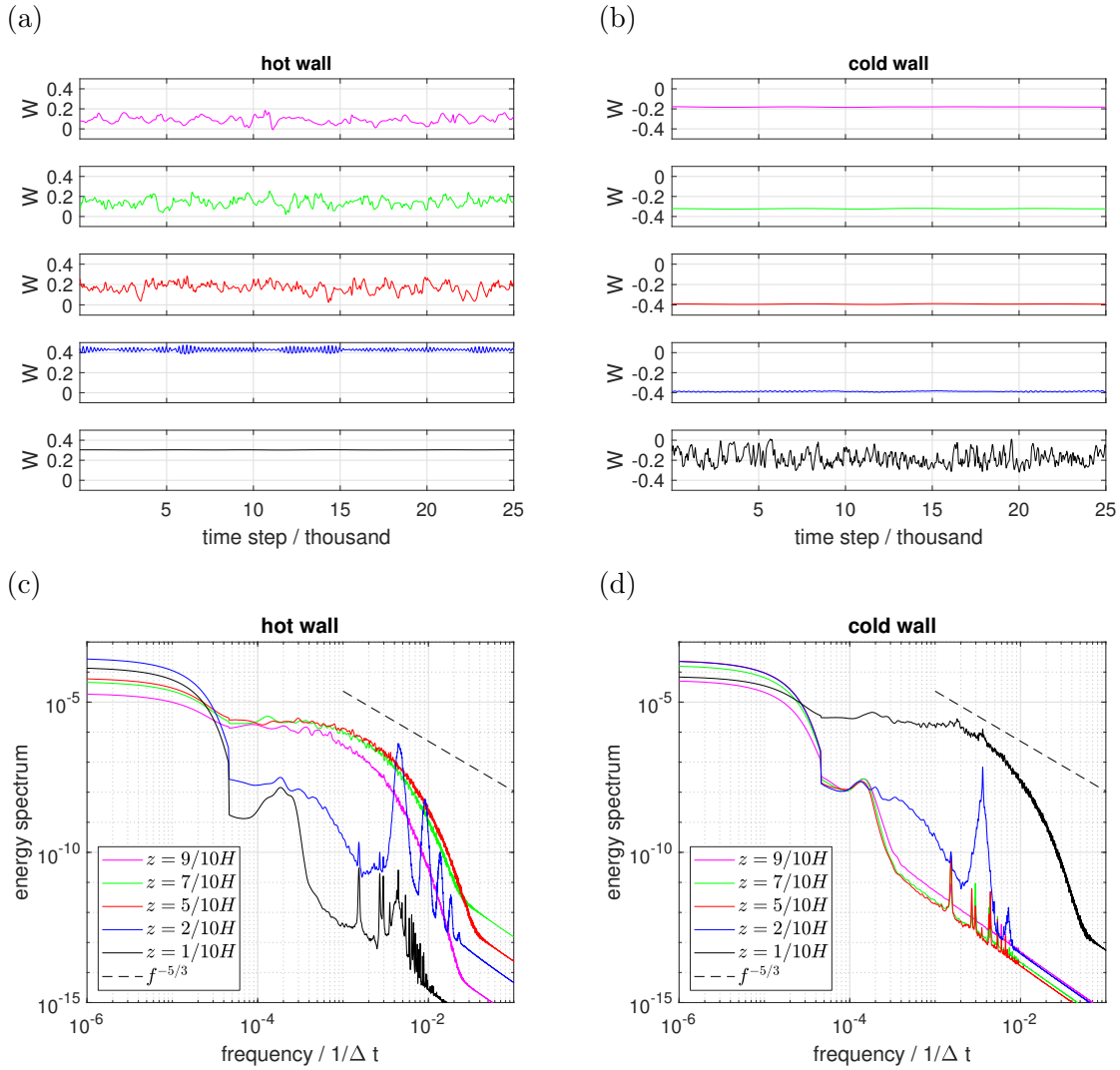


Figure 6.28: Results of boundary layer stability simulation: Sub-figures (a) and (b) show the velocity time series close to hot and cold walls, respectively. Sub-figures (c) and (d) show the corresponding spectra. The spectra are smoothed with a gaussian smoother to get a clearer representation.

and (d). On the hot wall the upper three time series contain much energy over a wide range of frequencies, as is expected in turbulent flow. Around the frequency of $3 \cdot 10^{-3}/\Delta t$ the slope also matches the theoretical value of $-5/3$. It is worth noting though that due to the low Reynolds number, no extensive cascade is observed.

The spectrum of the point in the region of turbulent onset shows an interesting behavior by having several distinct peaks that exceed the energy of the turbulent flow locally. These peaks might be related to the turbulent onset. At the hight of this first measure point two-dimensional vortices are generated periodically, see Fig. 6.24. The peaks further follow a harmonic, by being two, three, four and five times the base frequency. The base frequency is about $4.4 \cdot 10^{-3}/\Delta t$ and the subsequent frequencies are $9.2 \cdot 10^{-3}/\Delta t$, $1.4 \cdot 10^{-2}/\Delta t$, $1.8 \cdot 10^{-2}/\Delta t$ and $2.3 \cdot 10^{-2}/\Delta t$.

On the cold wall only the lowest point shows the turbulent spectrum. The point in the transition region features one substantial frequency at $3.6 \cdot 10^{-3}/\Delta t$, which might be related to the turbulent onset. It is accompanied by a second distinct but much lower peak with twice the frequency, i.e. $7.2 \cdot 10^{-3}/\Delta t$.

The relation of these frequencies to turbulent onset can be checked by looking at a time series of the Q-criterion. Snapshots per 50 coarse time steps are compared. The frequency can be computed by counting the number of waves that appear over a large number of snapshots. Over 338 snapshots 76 waves appear on the hot and 61 on the cold wall. Hence, the frequencies are $4.5 \cdot 10^{-3}/\Delta t$ on the hot wall and $3.6 \cdot 10^{-3}/\Delta t$ on the cold wall. These frequencies coincide with the base frequencies found in the spectrum.

The points in the laminar jet show very low energy content. It is remarkable that the three spectra show similar peaks. These peaks can be attributed to sound waves generated by the turbulence. These peaks also follow a harmonic and are also visible on the hot wall. This underlines the interpretation as sound waves, because they are omnipresent. In the spectra in the transition region these frequencies are still visible, while in the turbulent regions they go under in larger turbulent energies. Two harmonics with base frequencies $1.5 \cdot 10^{-3}/\Delta t$ and $2.7 \cdot 10^{-3}/\Delta t$ are found with multiple overtones. The horizontal eigen frequency of the cavity at low temperature is $4 \cdot 10^{-3}/\Delta t$ and the vertical eigen frequency is $2 \cdot 10^{-3}/\Delta t$. Hence, the measured frequencies are on the order of magnitude of the theoretical eigen frequencies. The dynamics of the cavity are more complex than just linear waves, such that the frequencies are lower.

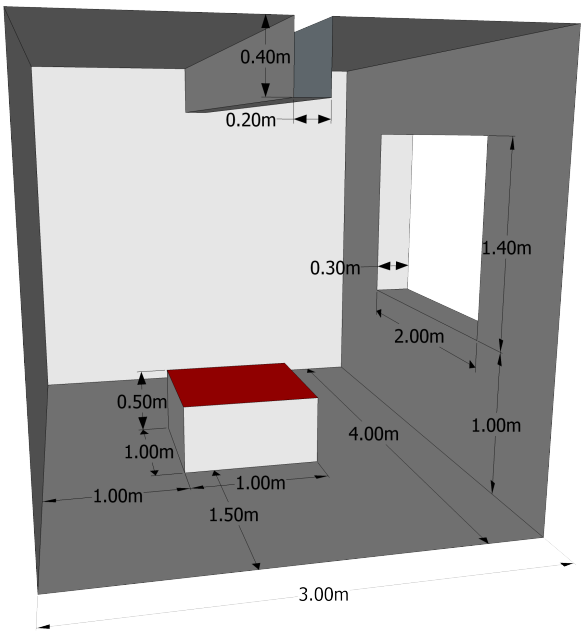
Albeit a more detailed analysis of the flow in the cavity with differentially heated walls is presented here, no concluding explanation could be identified. After this investigation the effects of cross flow disturbing the convective jets and decreasing acceleration along the jets are more likely explanations for turbulent onset, than the argument with diverging and converging flow at hot and cold walls that was proposed in [15]. Isolated test would have to be designed to gain more insight. It is not clear though, how these test would have to look like.

6.3.2 Simulation of a compartment fire

In this section VirtualFluidsGKS is applied to the simulation of a compartment fire. The long term aim of such simulations is, to predict fire loads on structural elements like beams and load carrying ceilings, in order to predict their durability under fire.

The compartment under investigation is shown in Fig. 6.29 (a). It is 4 m wide, 3 m deep, has height of 3 m and is ventilated by a single window, which spans over half the compartments width and has a height of 1.4 m. The ceiling is supported by a beam of intersection 0.2 m × 0.4 m. A burning obstruction of half a cubic meter is placed in the center of the compartment. The fuel mass flux boundary is only applied on top of the obstruction. The heat release rate is controlled by a measured heat

(a)



(b)

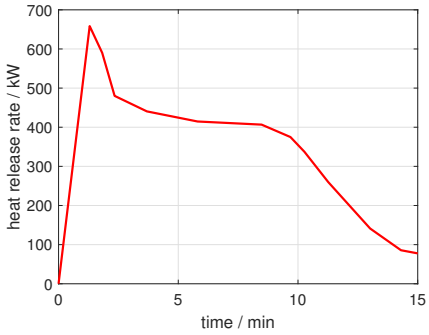


Figure 6.29: Compartment fire: Sub-figure (a) shows the dimensions of the compartment. On the red surface fuel is added. Sub-figure (b) shows the heat release rate curve applied for this fire. It is taken from [195].

release rate curve for the combustion of a wood pile taken from [195], see Fig. 6.29 (b). This setup describes a well ventilated fire, where enough fresh air is brought into the compartment to react all fuel inside the compartment. The numerical experiment is run for fifteen minutes.

The compartment is discretized by a non-uniform grid with a total of four levels. In the compartment the resolution is $\Delta x = 2.5\,\text{cm}$ with refinement in the plume and around the beam to 1.25 cm. Behind the window the domain is extended by 3.4 m and additional 2 m in height. Outside the compartment around the window the resolution is 5 cm and coarsens to the artificial boundaries that model the far field to 10 cm, see Fig. 6.30. The simulation comprises about 4.3 million cells on four GPUs. The top boundary is modeled with the advanced outflow boundary condition from Section 2.5.3 that was also used for the fire validation. The open boundary condition from Section 2.5.4 is applied to all outside boundaries, which are not walls.

For the modeling of the walls inside the compartment two approaches shall be compared here. First, the heat flux through the walls is neglected, such that the walls are insulated. Even though this might seem as a viable choice, physically the walls will take up heat by increasing their temperature. This effect is directly modeled in the second approach. Therein, the solid is also discretized and a heat equation is solved in the walls. For simplification solid heat transfer is only considered normal to the

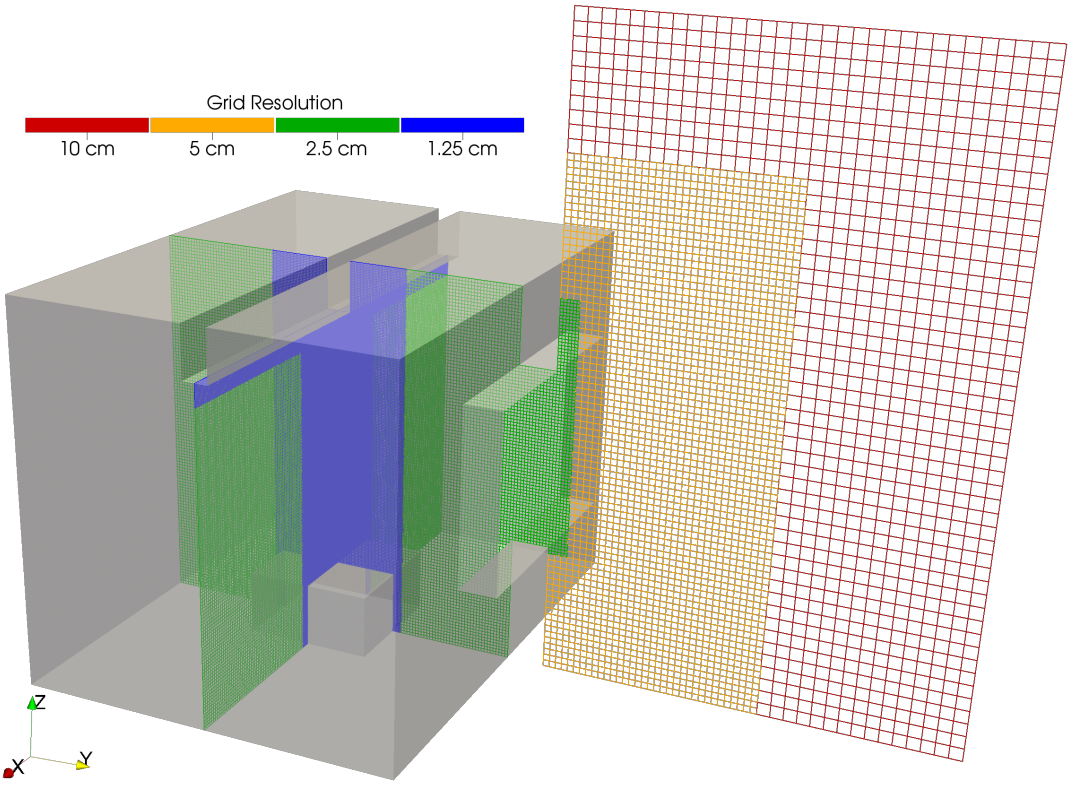


Figure 6.30: Compartment fire: The grid for the simulation has four levels with high resolution in the compartment and low resolution on the outside.

wall. For each cell on the boundary a solid domain that extends for 0.1 m inside the wall is used, disregarding the actual wall width and other geometrical constraints. At the end of the solid domain a Dirichlet boundary condition with ambient temperature is applied. This only holds under the assumption that the heat does not penetrate the walls during the time of the fire. On the fluid side a Dirichlet boundary condition with the temperature of the first adjacent fluid cell is used. The heat equation

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2} \tag{6.22}$$

is discretized with the FTCS (Forward in Time, Central in Space) scheme

$$T_i^{n+1} = T_i^n + k\Delta t\frac{T_{i+1}^n + T_{i-1}^n - 2T_i^n}{\Delta x^2}, \tag{6.23}$$

where n denotes the time level and i denotes the space enumeration of the points. The wall is discretized with a total of 64 internal points plus two points for the boundary conditions. Hence, the temperature is traced by the boundary condition at 64 points. The boundary condition for the fluid side is constructed as follows. The ghost cells are set by the insulated wall boundary condition, as in the first approach. Then the

heat flux into the solid is computed from the first finite difference points as

$$\mathcal{F}^{\rho E} = -\kappa \frac{-3T_0 + 4T_1 - T_2}{2\Delta x} \tag{6.24}$$

with the thermal conductivity $\kappa = k\rho C_p$ and the solid temperatures T_0 on the surface and T_1 and T_2 in the solid. The boundary condition is parametrized for concrete, such that $k = 10^{-6}\,\mathrm{m^2\,s^{-1}}$, $\rho = 2,400\,\mathrm{kg\,m^{-3}}$ and $C_p = 880\,\mathrm{J\,kg^{-1}\,K^{-1}}$. This boundary condition is applied to 23,376 cells, resulting in an additional 1.5 million degrees of freedom for the solid temperatures. The computational performance only drops slightly, due to the complex boundary condition from 189 MCUPS to 182 MCUPS.

The choice of using the first cells temperature as the wall temperature is not optimal. Fig. D.14 shows the solid temperature field at about five minutes. It is evident that the boundary condition does not correctly treat the refinement. On the finer level, the temperature is substantially smaller, which is probably due to the inaccurate temperate estimate on the wall. The figure further shows, how the temperature intrudes the solid without penetrating it, justifying the assumption of ambient temperature as second boundary condition.

The resulting compartment fires are shown in Figs. 6.31 and 6.32 for insulated and conducting walls, respectively. The visualization in these figures is tuned for visual appearance, rather than physical correctness. The pictures were rendered in Paraview [144]. The light emitting "fire" visualizes the temperature field by volume rendering. The color scheme is *erdc_color_BW*, which varies the color between black and bright yellow with orange in between. The color scheme is scaled between 900 K and 1300 K. Further, the opacity is linearly increased from 0 to 1 over the same range, hence, making low temperatures transparent and high temperatures opaque with a bright yellow. The visual "smoke" is a visualization of the combustion products Y_P in the range from 0.1 to 0.5. The color scheme is varied from a 50% grey to black and the opacity from 0 to 0.5. Physically, the light emitting flame would have to be computed by a radiation law of temperature and depending on the sooth content. Since, sooth is not considered here, this is not done. Further, the smoke thickness is not calibrated.

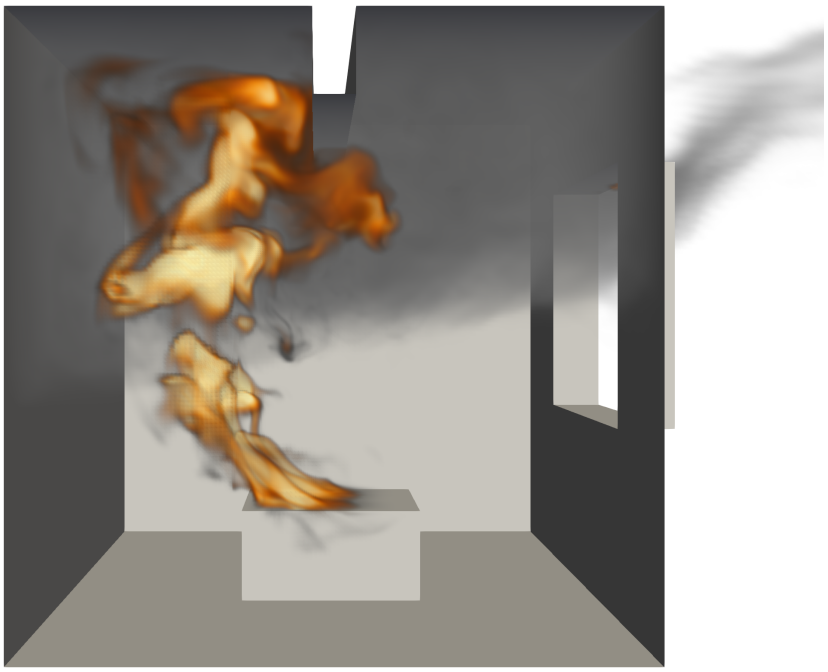
The flame evolution with insulated walls over the first two and a half minutes is available on YouTube under

`https://youtu.be/qsdd30cuSHM`.

In the video as well as in Fig. 6.31 (a) it is evident that the plume is shifted away from the window. This is due to fresh air entering the compartment through the lower half of the window, while hot gases leave the compartment through upper half.

Comparing the fire with the two different boundaries in Figs. 6.31 and 6.32 reveals a major difference. In the case with the insulated walls, there is no cooling in the compartment, other than the fresh air that is dragged into the flame and which is much heavier than the hot combustion products. It, hence, remains at the floor. This

(a)



(b)

Figure 6.31: Compartment fire: Fire visualization for the simulation with insulated walls at 01 : 30 min. Side view (a) and front view from window (b). The fresh air on the bottom and the smoke at the ceiling form two clear zones. The visualization is tuned for visual appearance.

(a)

(b)

Figure 6.32: Compartment fire: Fire visualization for the simulation with conducting walls at 01 : 30 min. Side view (a) and front view from window (b). Due to cooling at the walls smoke descends, such that the whole compartment is filled with smoke. The visualization is tuned for visual appearance.

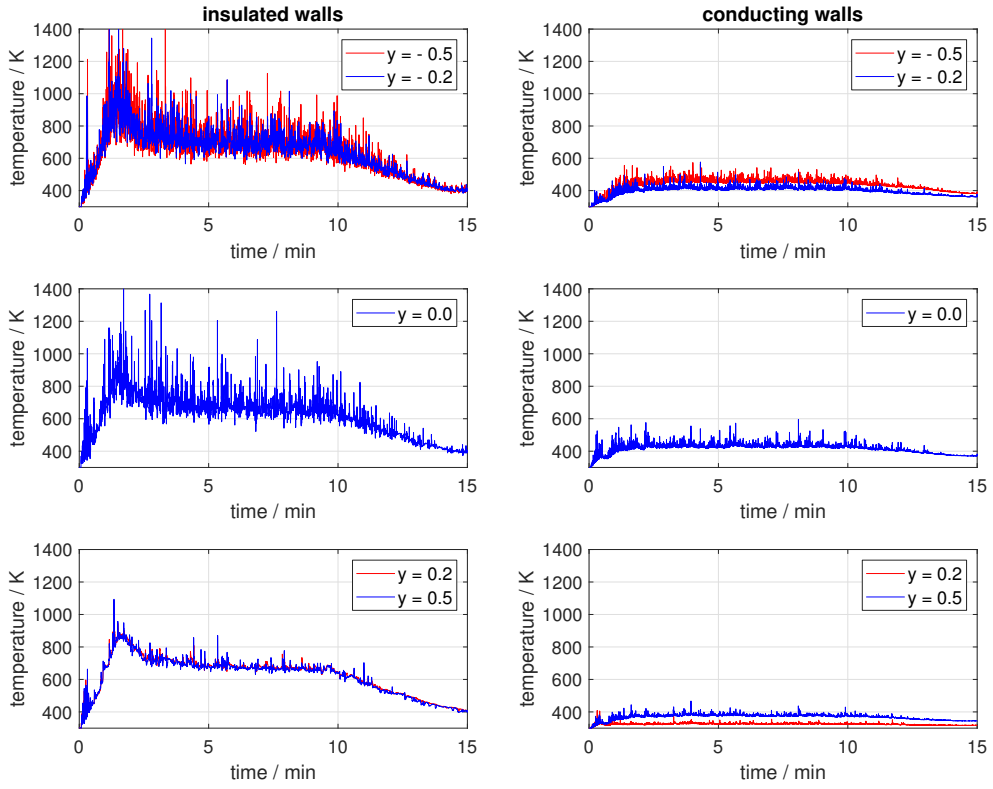


Figure 6.33: Compartment fire: Time series of velocity at five points on the ceiling. The y-coordinate goes from the back of the room to the window and the beam is around $y = 0$.

leads to the formation of two clear zones, one with combustion products and smoke in the upper half of the compartment and a second with cold fresh air at the bottom, see Fig. 6.31. Enabling heat conduction into the walls, cools the combustion products, such that smoke descends towards the floor, see Fig. 6.32. In this case no two zones can be identified.

For further comparison temperature time series at several points on the ceiling were recorded. Temperatures along the centerline of the room normal to the beam are shown in Fig. 6.33. The first two points are in the rear of the room (viewed from the window), the third in the middle below the beam and hence directly over the fire and the last two points towards the window. With adiabatic walls the temperature at the ceiling rises fast to about a thousand Kelvin with large oscillations and drops due to drop in heat release rate, see Fig. 6.29 (b). Towards the window the oscillations are weaker, this is due to the inflowing air, shifting the flame to the rear of the room. The temperature at all points is about 700 K. With the conducting walls the temperatures as well as the oscillations are much smaller. The wall temperature does not increase above 600 K. Also, the different points differ in temperature. The point in the back of the room has the highest temperature because the plume hits the ceiling

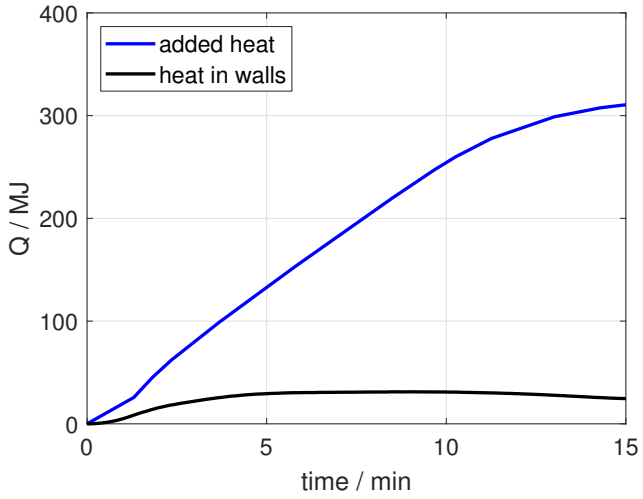


Figure 6.34: Compartment fire: Time series of the added heat and the heat stored in the walls.

a this location. In the front of the room, the point closer to the beam shows a lower temperature than further towards the window. Closer to the beam, the beam as well as the ceiling draw heat from the fluid.

As a final analysis the heat taken up by the walls is compared to the heat entered into the system by the fuel. The total amount of heat is computed by integrating the heat release rate curve (see Fig. 6.29 (b)) over time. The heat held by the solid is computed as

$$Q = \rho C_P \int_S T dS, \tag{6.25}$$

where S is the solid domain. In fifteen minutes about 300 MJ are added to the system, see Fig. 6.34. Ten percent of this heat is stored in the walls. After ten minutes the heat release rate drops, because most of the fuel is used up. From this point onwards, the walls emit heat to the room.

In this application case the applicability of VirtualFluidsGKS for realistic fire simulation is demonstrated. It is shown that the boundary conditions have a large influence on the flow evolution because the amount of heat taken up and stored by the walls is not negligible.

7 Conclusion and outlook

In the following two sections this work is concluded, the key findings are highlighted and an outlook for future development of VirtualFluidsGKS towards full applicability to fire simulation is given.

7.1 Conclusion

This work investigates the efficient implementation of GKS on nested Cartesian grids for massively parallel hardware in terms of multiple GPUs. The applicability of the present GKS implementation is tested and validated for the simulation of turbulent natural convection and fire.

To this end, first the relevant gas kinetic theory necessary for the derivation and understanding of GKS is introduced. An extensive literature review on the history and variants of GKS is then given. This allows us to put the present GKS in a context. The present GKS is then derived in detail. The resulting scheme is a simplification for low Mach number flows of the original GKS, which considers high Mach numbers. Its realization on uniform Cartesian grid is presented subsequently including boundary conditions.

For reasons of performance and automated grid generation, the finite volume method in this work is implemented on locally uniform Cartesian grids. Refinement is applied by coupling grids of different resolution. A novel algorithm for this coupling is proposed and tested. This coupling is second-order accurate, conservative, has low pressure reflections and is general in the sense that no special cases for complex interface topologies have to be considered. Further, the refinement with integer refinement ratios (where only a ratio of two is considered here) allows for the performance enhancement obtained from nested time stepping.

For the modeling of fire, first a brief overview of available methods is given. Then a simple combustion model for non-premixed flames with instantaneous reaction and turbulence based mixing is chosen and the relevant equations are given. It was found that this simple model has several stability issues which were fixed by the introduction of suitable limiters.

After introducing the theory behind GKS for simulation of turbulent natural convection and fire, the implementation of VirtualFluidsGKS is presented. The code was developed bottom up from scratch. The implementation features CPU/GPU dualism that allows execution of computations on either CPU or GPU, depending on the choice

of the user. This dualism is implemented by a strategy pattern on the data side, where two allocator classes handle memory management for CPU and GPU, respectively. On the algorithm side a template function is used to either iterate over computational entities (cell faces or cells) or distribute the work on the GPU by a kernel function. The GPU implementation is based on the NVIDIA CUDA framework. The performance of VirtualFluidsGKS is tested and high update rates of up to 300 million cell updates per second on a professional GPU are shown. The CPU performance on a standard desktop CPU is lower by two orders of magnitude. Since the scalability of single GPU computations is limited by limited memory and bandwidth, Multi-GPU communication is also considered. A block domain decomposition is implemented with sequential communication in the three coordinate directions. The Multi-GPU implementation features communication hiding, where GPU computations are performed concurrently to GPU memory operations and data transfer between processes. While it is found that the communication hiding has small negative performance implications at low numbers of GPUs (up to 4 GPUs), it brings substantial improvements in parallel efficiency at larger numbers of GPUs (tested up to 32 GPUs). The chosen block domain decomposition is suitable for lower numbers of GPUs, where a trivial load balancing is possible. For very complex flow domains with multiple unsymmetrical refinements and distribution to many GPUs (e.g. hundreds) this decomposition is not efficient. This is because the refinement levels are distributed based on a geometric decomposition of the whole grid. Hence, there is no guarantee or control that the load (which is mostly dominated by the finest levels) is evenly distributed, because most fine cells will end up on a few GPUs. In order to scale to large complex systems, it is necessary to distribute grid levels independently, such that each process has a similar amount of cells on each level. This would require communication on the grid interface as implemented in VirtualFluidsCPU.

Algorithms for the automated and distributed generation of grids for GKS and LBM (i.e. VirtualFluidsGKS and VirtualFluidsGPU) are presented. Based on this grid generation a novel morph-cell approach in two dimensions for complex boundaries in the finite volume method is proposed. The idea behind this approach is to use the sub-grid distances used in LBM boundary conditions to adapt the grid to the boundary. The applicability of this approach is tested on inclined channel flow and two-dimensional flow around a cylinder. Two reconstructions are compared. It is found that average interpolation is more stable but introduces large pressure errors. Extrapolation on the other hand suffers stability issues, but shows second-order convergence and substantially less errors in pressure. An extension of this algorithm to three dimensions is not considered.

For the validation of VirtualFluidsGKS several test cases were performed. The standard benchmark of the square cavity with differentially heated walls at large temperature difference shows the second order of accuracy for laminar natural convection. The two-dimensional Rayleigh-Bénard test, shows the validity also for turbulent natural convection in two dimensions. The validity for three-dimensional turbulence is shown on the decay of a Taylor-Green vortex. The GKS results are not as good as results obtained by the fourth-order convergent LBM that is used in VirtualFluidsGPU and VirtualFluidsCPU, but the results are still acceptable. GKS shows

numerical dissipation at fine scales, reducing the measured enstrophy. It is found that this effect does not only depend on spatial resolution, but also on time step size. Three dimensional turbulent convection is validated against experimental data and compared to other simulations of a three dimensional cavity flow at small temperature difference. The GKS results agree well with experiments and show similar agreement as the reference simulation. While GKS shows slightly worse agreement in the temperature profiles, it shows better agreement in velocity fluctuations.

Finally, two validation cases involving fire are investigated and results are compared to experimental data and results obtained by FDS. First, it is found that the VirtualFluidsGKS results depend more on spatial resolution then the FDS results. The accuracy is still found to be acceptable for higher resolutions. Additionally, the time to solution was compared between FDS and VirtualFluidsGKS. The latter is found to be more than an order of magnitude faster due to the efficient GPU implementation.

After showing the validity of VirtualFluidsGKS, it is applied to two flow problems. First, an investigation of boundary layer stability of natural convection in cavities with differentially heated walls is continued from a previous two-dimensional study. The asymmetry in turbulence onset is analyzed and two mechanism that possibly drive the transition are identified. First, a strong horizontal flow from the cold to the hot wall is found that impinges on the hot wall at the location where the convective jets transitions to turbulence. Second, the turbulence starts where the convective jet enters a region, where the surrounding density is similar to the jet density. At this point the buoyancy acceleration drops. A concluding explanation was not obtained, as that would require isolated tests to analyze these phenomena.

The second application investigates a well ventilated compartment fire. Apart from showing that this can be simulated, two different boundary conditions with insulated and conducting walls are compared. It is found that the heat uptake of the walls is on the order of ten percent of the heat added by the fire and is, hence, not negligible. This cooling effect leads to combustion products being dragged down towards the floor of the compartment, filling the lower parts of the room with smoke. This does not happen with insulated walls, where the bottom is free from combustion products.

Concluding, this work lays the foundation for the application of GKS for the simulation of fire. With several test cases it is shown that GKS is a viable candidate for high performance simulation of thermal compressible flow and fire.

7.2 Outlook

Even after showing the validity of VirtualFluidsGKS for fire simulation, we have to note that its applicability is currently limited. Several points will have to be addressed to improve its applicability.

Concerning the flow solver the topic of complex boundaries was tackled with the proposition of the morph-cell algorithm. While this algorithm produces a good boundary representation of the grid, it struggles with the numerics, especially in terms of stability. Further, it is not clear at this point whether and how this approach is extensible to three dimensions. Further work of VirtualFluidsGKS will have to deal with this topic of complex boundaries, either by enhancing the morph-cell approach, or by implementing well established cut-cell approaches for the finite volume method.

In terms of turbulence simulation only the elementary static Smagorinsky model was considered in this work. In the field of LES other turbulence models are known to perform better. This will have to be investigated in the future.

The combustion model for fire in this work is very simple. More accurate models are available, for example in FDS. The lower accuracy of VirtualFluidsGKS compared to FDS for the fire benchmarks is probably partially due to the simple model. In the future better combustion models have to be considered.

Finally, the modeling range of VirtualFluidsGKS is much smaller than that of FDS. The physically non-negligible effect of thermal radiation is not considered at all. Solid heat transfer was only considered in a small example. With regard to fire, pyrolysis modeling is missing. It is required to investigate not only clearly quantified fires, but also the spread of fires. In this work the heat release rate of the fires is directly specified. In a real fire it is determined by the combustion of burnable materials and their ignition. For this pyrolysis modeling is mandatory.

VirtualFluidsGKS has proven its capabilities and provides an extensible framework to develope, implement and test the missing features. The present implementation of VirtualFluidsGKS builds the foundation for high-performance fire simulation.

A Additional Derivations

In this appendix some detailed symbolic calculation are given that were left out in the derivations in the main text.

A.1 Derivation of expansion coefficients

$$\begin{aligned} A_5 = -2\frac{\partial\lambda}{\partial t} = & -2\frac{\partial}{\partial t}\left(\frac{K+3}{4}\left(E-\frac{1}{2}\vec{U}^2\right)^{-1}\right) \\ = & \quad 2\frac{\partial}{\partial t}\left(\frac{K+3}{4}\left(\frac{\partial E}{\partial t}-\vec{U}\cdot\frac{\partial\vec{U}}{\partial t}\right)\left(E-\frac{1}{2}\vec{U}^2\right)^{-2}\right) \\ = & \quad 2\frac{\partial}{\partial t}\left(\frac{K+3}{4}\left(\frac{\partial E}{\partial t}-\vec{U}\cdot\frac{\partial\vec{U}}{\partial t}\right)\left(\frac{K+3}{4\lambda}\right)^{-2}\right) \\ = & \quad 2\frac{4\lambda^2}{K+3}\left(\frac{\partial E}{\partial t}-\vec{U}\cdot\frac{\partial\vec{U}}{\partial t}\right) \end{aligned} \tag{A.1}$$

$$\begin{aligned} A_2 = 2\frac{\partial\lambda U}{\partial t} &= 2\lambda\frac{\partial U}{\partial t} - A_5 U \\ A_3 = 2\frac{\partial\lambda V}{\partial t} &= 2\lambda\frac{\partial V}{\partial t} - A_5 V \\ A_4 = 2\frac{\partial\lambda W}{\partial t} &= 2\lambda\frac{\partial W}{\partial t} - A_5 W \end{aligned} \tag{A.2}$$

$$
\begin{aligned}
A_1 &= \frac{\partial}{\partial t}\left(\ln\rho + \frac{K+3}{2}\ln\left(\frac{\lambda}{\pi}\right) - \lambda\vec{U}^2\right) \\
&= \left(\frac{1}{\rho}\frac{\partial\rho}{\partial t}\right) + \frac{K+3}{2\lambda}\frac{\partial\lambda}{\partial t} - \frac{\partial\lambda}{\partial t}\vec{U}^2 - 2\lambda\frac{\partial\vec{U}}{\partial t}\cdot\vec{U} \\
&= \left(\frac{1}{\rho}\frac{\partial\rho}{\partial t}\right) + \frac{K+3}{2\lambda}\frac{\partial\lambda}{\partial t} - \frac{\partial\lambda}{\partial t}\vec{U}^2 - 2\frac{\partial\lambda\vec{U}}{\partial t}\cdot\vec{U} + 2\frac{\partial\lambda}{\partial t}\vec{U}^2 \\
&= \left(\frac{1}{\rho}\frac{\partial\rho}{\partial t}\right) + \frac{K+3}{2\lambda}\frac{\partial\lambda}{\partial t} + \frac{\partial\lambda}{\partial t}\vec{U}^2 - 2\frac{\partial\lambda\vec{U}}{\partial t}\cdot\vec{U} \\
&= \left(\frac{1}{\rho}\frac{\partial\rho}{\partial t}\right) - 2\frac{\partial\lambda\vec{U}}{\partial t}\cdot\vec{U} + \left(\frac{1}{2}\vec{U}^2 + \frac{K+3}{4\lambda}\right)2\frac{\partial\lambda}{\partial t} \\
&= \left(\frac{1}{\rho}\frac{\partial\rho}{\partial t}\right) - A_2U - A_3V - A_4W - \left(\frac{1}{2}\vec{U}^2 + \frac{K+3}{4\lambda}\right)A_5
\end{aligned}
\tag{A.3}
$$

A.2 Integration of formal BGK solution

The formal solution of the BGK equation (see Eq. (2.83)) is

$$
f(\vec{0},\vec{u},t) = e^{-t/\tau}f(-\vec{u}t,\vec{u}-\vec{g}t,0) + \frac{1}{\tau}\int_0^t e^{-(t-t')/\tau}f^{eq}(-\vec{u}(t-t'),\vec{u}-\vec{g}(t-t'),t')dt'. \tag{A.4}
$$

The distributions f^{eq} and f therein are modeled as

$$
f^{eq}(\delta\vec{x},\vec{u}+\delta u,\delta t) = f_0^{eq}\left(1+\delta\vec{x}\cdot\vec{a}+\delta\vec{u}\cdot\vec{b}+\delta tA\right) \tag{A.5}
$$

and

$$
f(\delta\vec{x},\vec{u}+\delta u,0) = f_0^{eq}\left(1+\delta\vec{x}\cdot\vec{a}+\delta\vec{u}\cdot\vec{b}-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right). \tag{A.6}
$$

respectively, see Eqs. (2.85) and (2.97). The initial equilibrium distribution f_0^{eq} is independent of time (as it is defined for specific point in time). Inserting these into the BGK solution gives

$$
\begin{aligned}
f(\vec{0},\vec{u},t)/f_0^{eq} = \frac{1}{\tau}\int_0^t e^{-(t-t')/\tau}\left(1-\vec{u}\cdot\vec{a}(t-t')-\vec{g}\cdot\vec{b}(t-t')+At'\right)dt' \\
+ e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right).
\end{aligned}
\tag{A.7}
$$

The Integral can be solved with integration by parts

$$\begin{aligned} f(\vec{0},\vec{u},t)/f_0^{eq} = \frac{1}{\tau}\left[\tau e^{-(t-t')/\tau}\left(1-\vec{u}\cdot\vec{a}(t-t')-\vec{g}\cdot\vec{b}(t-t')+At'\right)\right]_0^t \\ -\frac{1}{\tau}\int_0^t \tau e^{-(t-t')/\tau}\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)dt' \\ +e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right) \end{aligned} \tag{A.8}$$

and the remaining integral can than be solved directly:

$$\begin{aligned} f(\vec{0},\vec{u},t)/f_0^{eq} = \frac{1}{\tau}\left[\tau e^{-(t-t')/\tau}\left(1-\vec{u}\cdot\vec{a}(t-t')-\vec{g}\cdot\vec{b}(t-t')+At'\right)\right]_0^t \\ -\frac{1}{\tau}\left[\tau^2 e^{-(t-t')/\tau}\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right]_0^t \\ +e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right). \end{aligned} \tag{A.9}$$

Then the limits of the integrals are applied to obtain

$$\begin{aligned} f(\vec{0},\vec{u},t)/f_0^{eq} = (1+At) - e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t)'\right) \\ -\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)+\tau e^{-t/\tau}\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right) \\ +e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right) \end{aligned} \tag{A.10}$$

and after sorting

$$\begin{aligned} f(\vec{0},\vec{u},t)/f_0^{eq} = (1+At) - \tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right) \\ -e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right) \\ +e^{-t/\tau}\left(1-\vec{u}\cdot\vec{a}\;t-\vec{g}\cdot\vec{b}\;t-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}+A\right)\right). \end{aligned} \tag{A.11}$$

The last terms cancel out and the time dependent distribution function on the interface is:

$$f(\vec{0},\vec{u},t) = f_0^{eq}\left(1-\tau\left(\vec{u}\cdot\vec{a}+\vec{g}\cdot\vec{b}\right)+(t-\tau)A\right). \tag{A.12}$$

B Interpolation weights for the grid interface

Apart from the argumentation of self consistent coarse to fine and fine to coarse interpolations in Section 2.7.3, the free weight can be obtained by minimizing the number of error terms.

The general interpolation stencil for the coarse to fine interpolation is formulated as

$$\underline{\tilde{W}}^{i,j,k}_{0,0,0} = w_0 \underline{W}_{0,0,0} + \sum_{\substack{l=i \vee \\ m=j \vee \\ n=k}} w_1 \underline{W}^{i,j,k}_{l,m,n} + \sum_{\substack{l=-i \vee \\ m=-j \vee \\ n=-k}} w_2 \underline{W}^{i,j,k}_{l,m,n}. \tag{B.1}$$

Inserting Taylor expansion and eliminating first order terms yields the weights

$$w_1 = \frac{7}{24} - \frac{w_0}{6} \qquad \text{and} \qquad w_2 = \frac{1}{24} - \frac{w_0}{6}. \tag{B.2}$$

The weight w_0 remains undetermined and the accuracy of the interpolation is

$$\begin{aligned} \underline{\tilde{W}}_{0,0,0} = & \underline{W}_{0,0,0} \\ & - (16w_0 - 13)\frac{\Delta x^2}{96}\left(\frac{\partial^2}{\partial x^2}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial y^2}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial z^2}\underline{W}_{0,0,0}\right) \\ & - \frac{\Delta x^2}{16}\left(\frac{\partial^2}{\partial xy}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial xz}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial yz}\underline{W}_{0,0,0}\right) + \mathcal{O}(\Delta x^3). \end{aligned} \tag{B.3}$$

The mixed derivatives cannot be modified by the choice of w_0. The second unidirectional derivatives can be eliminated by choosing

$$w_0 = \frac{13}{16}, \tag{B.4}$$

Such that the accuracy of the interpolation is

$$\underline{\tilde{W}}_{0,0,0} = \underline{W}_{0,0,0} - \frac{\Delta x^2}{16}\left(\frac{\partial^2}{\partial xy}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial xz}\underline{W}_{0,0,0} + \frac{\partial^2}{\partial yz}\underline{W}_{0,0,0}\right) + \mathcal{O}(\Delta x^3). \tag{B.5}$$

C Least-square solution of over determined systems of linear equations

We consider the generic equation

$$\underline{\underline{A}}\ \underline{x} = \underline{b}, \tag{C.1}$$

where $\underline{\underline{A}} \in \mathbb{R}^{m\times n}$, $\underline{x} \in \mathbb{R}^n$, $\underline{b} \in \mathbb{R}^m$ and $m \geq n$, such that the equation is over determined.

C.1 QR-decomposition

First $\underline{\underline{A}}$ is decomposed in the orthogonal matrix $\underline{\underline{Q}} \in \mathbb{R}^{m\times n}$ and the upper triangular matrix $\underline{\underline{R}} \in \mathbb{R}^{n\times n}$ such that

$$\underline{\underline{A}} = \underline{\underline{Q}}\ \underline{\underline{R}}. \tag{C.2}$$

After the decomposition, a least square solution of an over-determined linear system $\underline{\underline{A}}\ \underline{x} = \underline{b}$, with $\underline{x} \in \mathbb{R}^n$ and $\underline{b} \in \mathbb{R}^m$ is straight forward

$$\underline{\underline{Q}}\ \underline{\underline{R}}\ \underline{x} = \underline{b} \qquad \leftrightarrow \qquad \underline{\underline{R}}\ \underline{x} = \underline{\underline{Q}}^T\ \underline{b}. \tag{C.3}$$

The orthogonal property of $\underline{\underline{Q}}$ allows trivial inversion and $\underline{\underline{R}}$ being an upper triangular matrix allows backward insertion to solve for $\underline{x}$.

C.2 Gram-Schmidt orthogonalization

The decomposition of $\underline{\underline{A}}$ is

$$\underline{\underline{A}} = [\underline{a}_1 \cdots \underline{a}_n] = \left[\underline{q}_1 \cdots \underline{q}_n\right] \begin{bmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \end{bmatrix}. \tag{C.4}$$

Solving for a single vector $\underline{q}_j$ by backward insertion yields

$$\underline{q}_j = \frac{1}{r_{jj}} \left(\underline{a}_j - \sum_{k=1}^{j-1} r_{kj} \underline{q}_k \right), \tag{C.5}$$

where r_{ik} remains undetermined.

Further, we know that the vectors $\underline{q}_j$ must be orthogonal. This can be ensured by constructing them based on the Gram-Schmidt process [196, Chapter 1.3.2] of the vectors $\underline{a}_j$, such that

$$\underline{q}_j = \frac{\underline{q}'_j}{\|\underline{q}'_j\|} \qquad \text{with} \qquad \underline{q}'_j = \underline{a}_j - \sum_{k=1}^{j-1} (\underline{q}_k \cdot \underline{a}_j) \underline{q}_k. \tag{C.6}$$

Equating coefficients between Eqs. (C.5) and (C.6) yields

$$r_{ij} = \underline{q}_i \cdot \underline{a}_j \quad \text{for} \quad i < j \tag{C.7}$$

and

$$r_{ij} = \|\underline{q}'_j\| = \left\| \underline{a}_i - \sum_{k=1}^{i-1} (\underline{q}_i \cdot \underline{a}_j) \underline{q}_k \right\| = \left\| \underline{a}_i - \sum_{k=1}^{i-1} r_{kj} \underline{q}_k \right\| \quad \text{for} \quad i = j. \tag{C.8}$$

The latter expression requires the computation of the magnitude of the vector, and hence the computation of its square. Many terms in the result are zero, though because of the orthogonality

$$\underline{q}_i \cdot \underline{q}_j = 0 \quad \text{for} \quad i \neq j \tag{C.9}$$

and the normalization yields

$$\underline{q}_i \cdot \underline{q}_j = 1 \quad \text{for} \quad i = j. \tag{C.10}$$

With these simplifications, the diagonal elements are

$$r_{ij} = \sqrt{\underline{a}_i \cdot \underline{a}_i - \sum_{k=1}^{i-1} r_{kj}^2}, \quad i = j \tag{C.11}$$

Hence, the decomposition can be computed successively as

$$r_{ij} = \begin{cases} \underline{q}_i \cdot \underline{a}_j & , \quad i < j \\ \sqrt{\underline{a}_i \cdot \underline{a}_i - \sum_{k=1}^{i-1} r_{kj}^2} & , \quad i = j \\ 0 & , \quad i > j \end{cases} \qquad \text{and} \qquad \underline{q}_j = \frac{1}{r_{jj}} \left(\underline{a}_j - \sum_{k=1}^{j-1} r_{kj} \underline{q}_k \right). \tag{C.12}$$

C.3 Solution

Finally, the backward insertion for the solution of $\underline{x}$ is

$$x_i = \frac{1}{r_{ii}} \left(\underline{q}_i \cdot \underline{b} - \sum_{k=i+1}^{n} r_{ik} x_k \right). \tag{C.13}$$

C.4 Efficient calculation of the QR decomposition

The previous algorithm goes through the matrix column wise. It is beneficial, though, to go through it row wise instead.

First all the projections are computed as

$$r'_{ij} = \underline{a}_i \cdot \underline{a}_j \ , \quad i \leq j. \tag{C.14}$$

Now, we insert the definition of q_j into r_{ij} and obtain for $i \neq j$

$$r_{ij} = \underline{q}_i \cdot \underline{a}_j = \frac{1}{r_{ii}} \left(\underline{a}_i - \sum_{k=1}^{i-1} r_{ki} \underline{q}_k \right) \cdot \underline{a}_j. \tag{C.15}$$

Solving the dot product and using $\underline{q}_k \cdot \underline{a}_j$ we obtain

$$r_{ij} = \underline{q}_i \cdot \underline{a}_j = \frac{1}{r_{ii}} \left(\underline{a}_i \cdot \underline{a}_j - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right). \tag{C.16}$$

This can be done in a single loop over the rows of $\underline{\underline{A}}$. The elements of $\underline{\underline{R}}$ can be computed in place (just dropping the prime)

$$r_{ij} = \begin{cases} \frac{1}{r_{ii}} \left(r'_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right) & , \quad i < j \\ \sqrt{r'_{ij} - \sum_{k=1}^{i-1} r_{kj}^2} & , \quad i = j \end{cases}. \tag{C.17}$$

This makes the computation of $\underline{\underline{R}}$ independent of explicitly computing $\underline{\underline{Q}}$. The $\underline{\underline{Q}}$ is not required to stored, but can be computed directly when needed.

D Additional Results

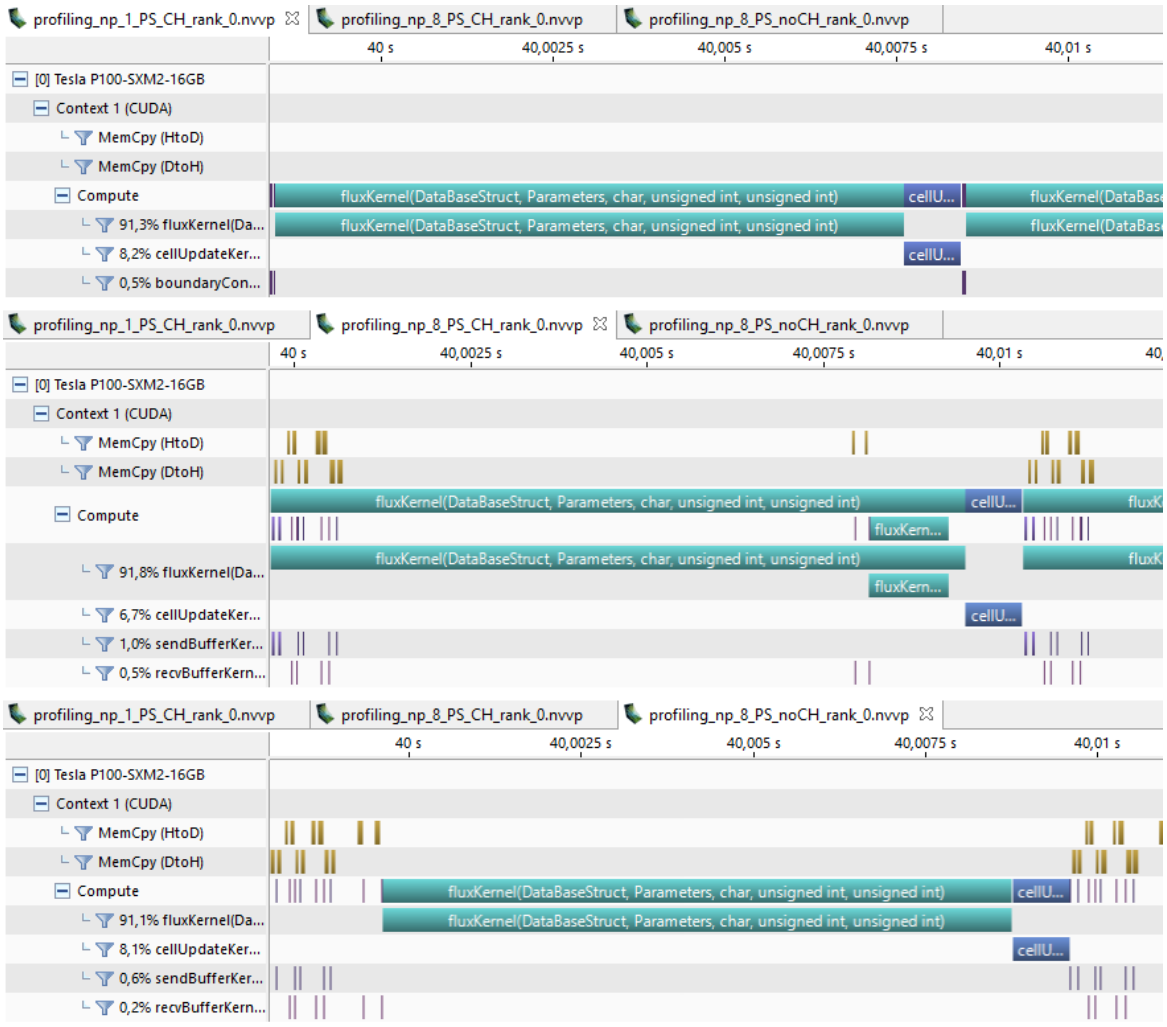


Figure D.1: Communication hiding profiling results: From top to bottom: Single GPU, Multi GPU with communication hiding, Multi GPU without communication hiding. This test was performed with 8 GPUs under weak scaling, i.e. with 128^3 cells per GPU. The time lines are synchronized at the beginning of a time step. It is clearly visible, that the communication is interleaved with the computation, when communication hiding is used. The single GPU run time is 70 s. Communication reduces reduces the run time to 77 s and 81 s with and without communication hiding, respectively.

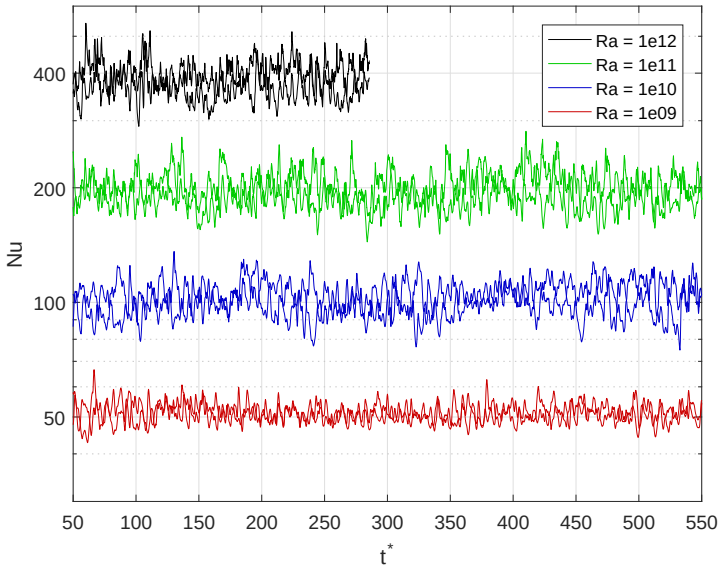


Figure D.2: Rayleigh-Bénard convection at high Rayleigh number. Figure reused from [97]

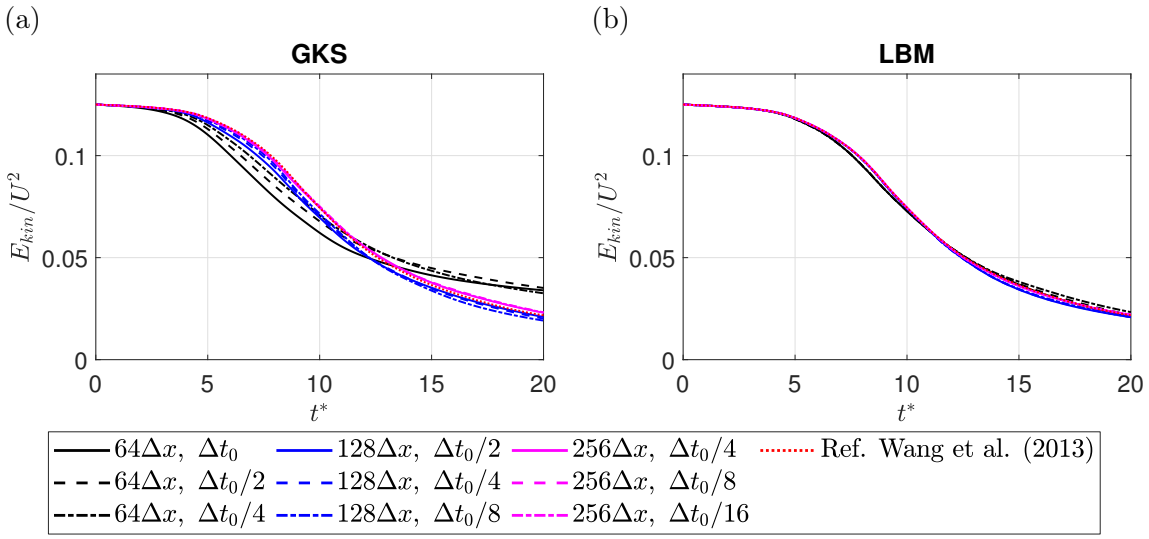


Figure D.3: Integral kinetic energy and enstrophy for Taylor-Green vortex at $Re = 1600$ compared to spectral result by Wang et al. [177]. The LBM solution is obtained with fourth order convergent LBM [180, 181, 182].

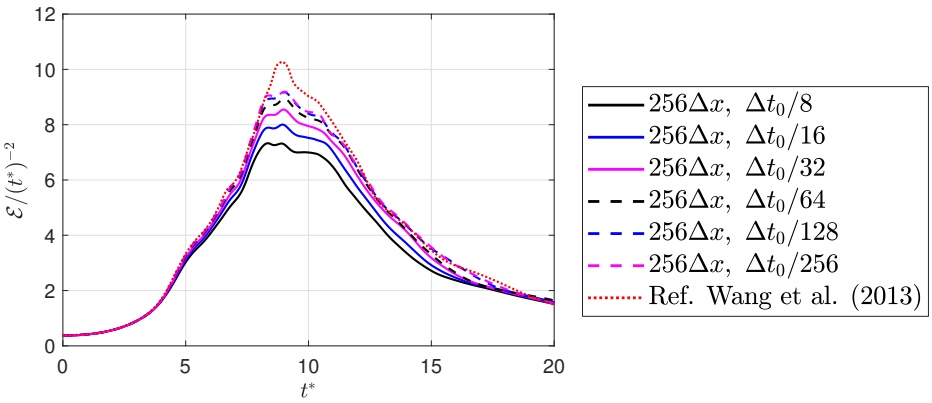


Figure D.4: Taylor-Green vortex: influence of time step

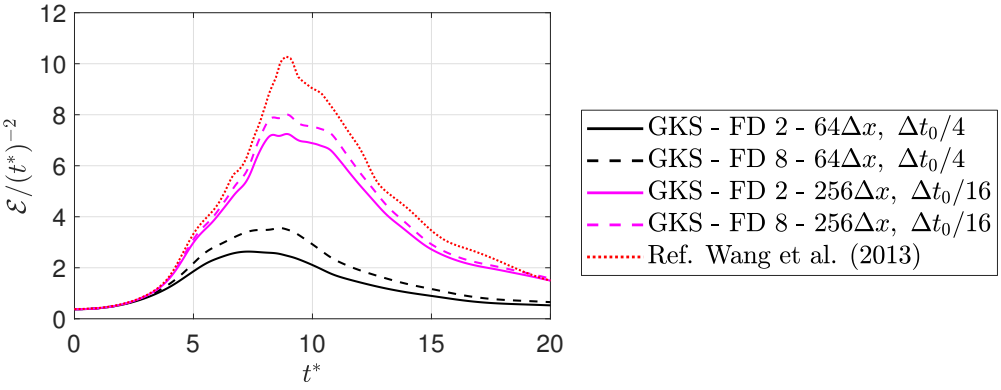


Figure D.5: Taylor-Green vortex: influence of finite difference order

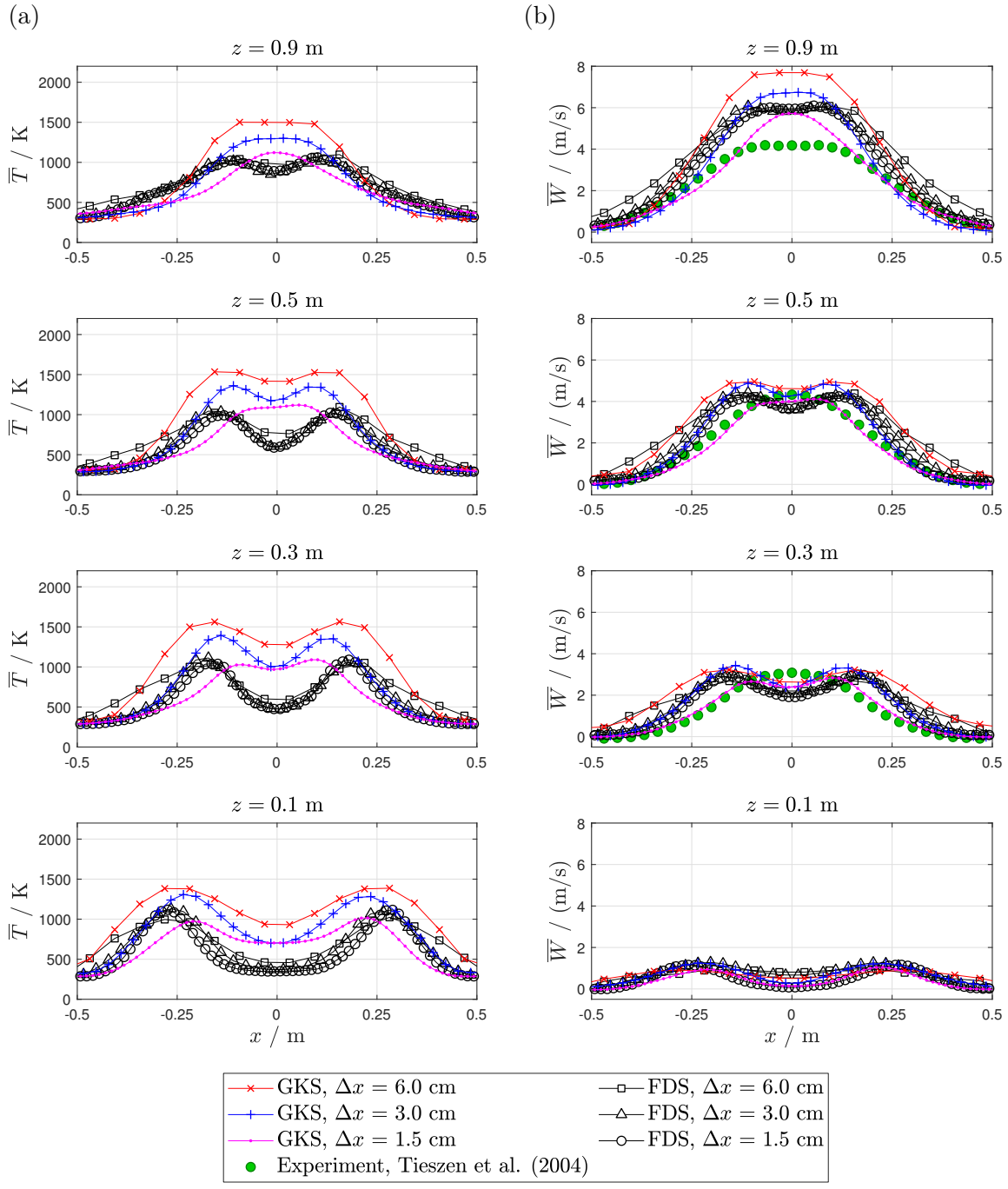


Figure D.6: Sandia flame, Test 14: temperature and vertical velocity profiles

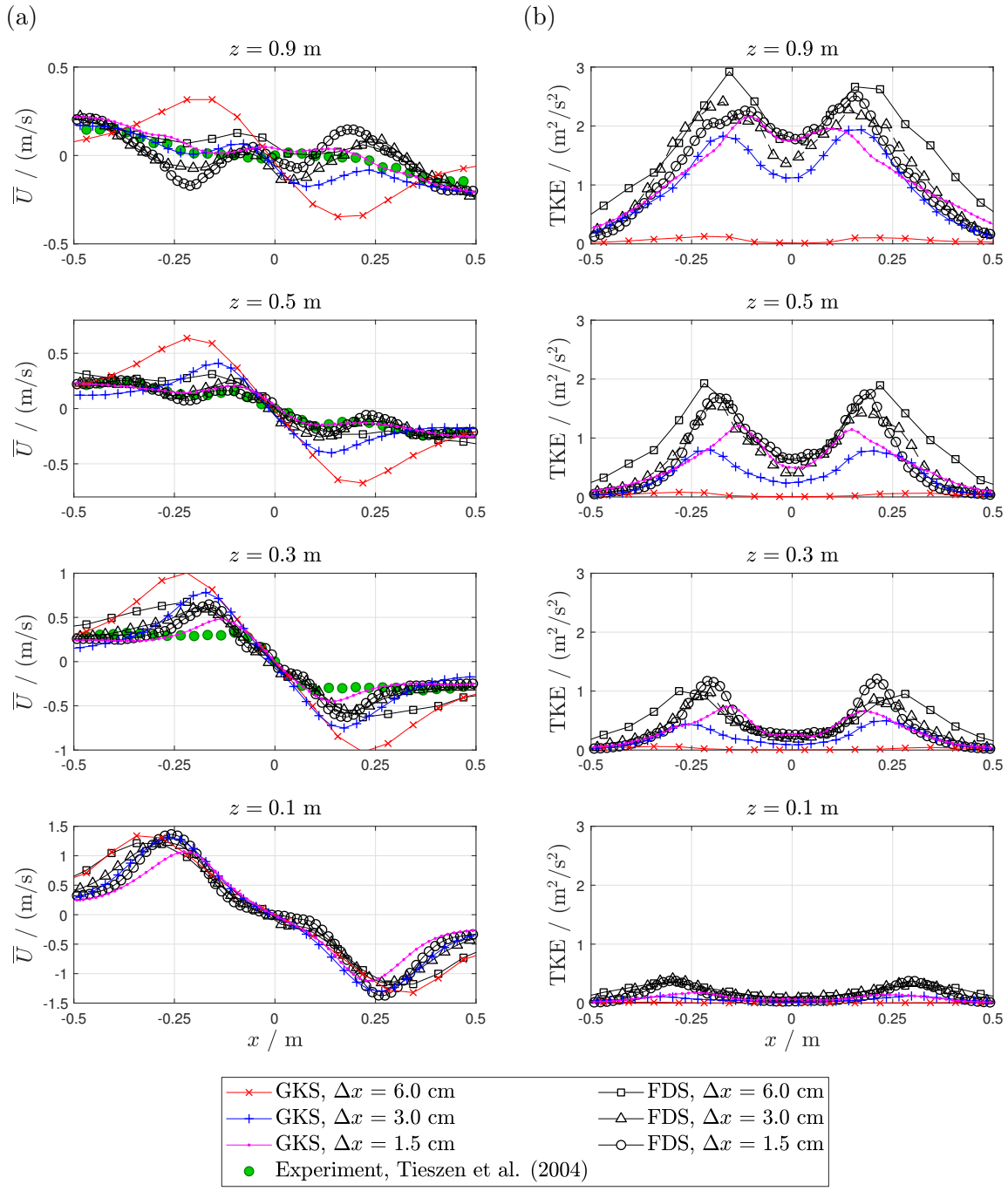


Figure D.7: Sandia flame, Test 14: horizontal velocity and turbulent kinetic energy profiles

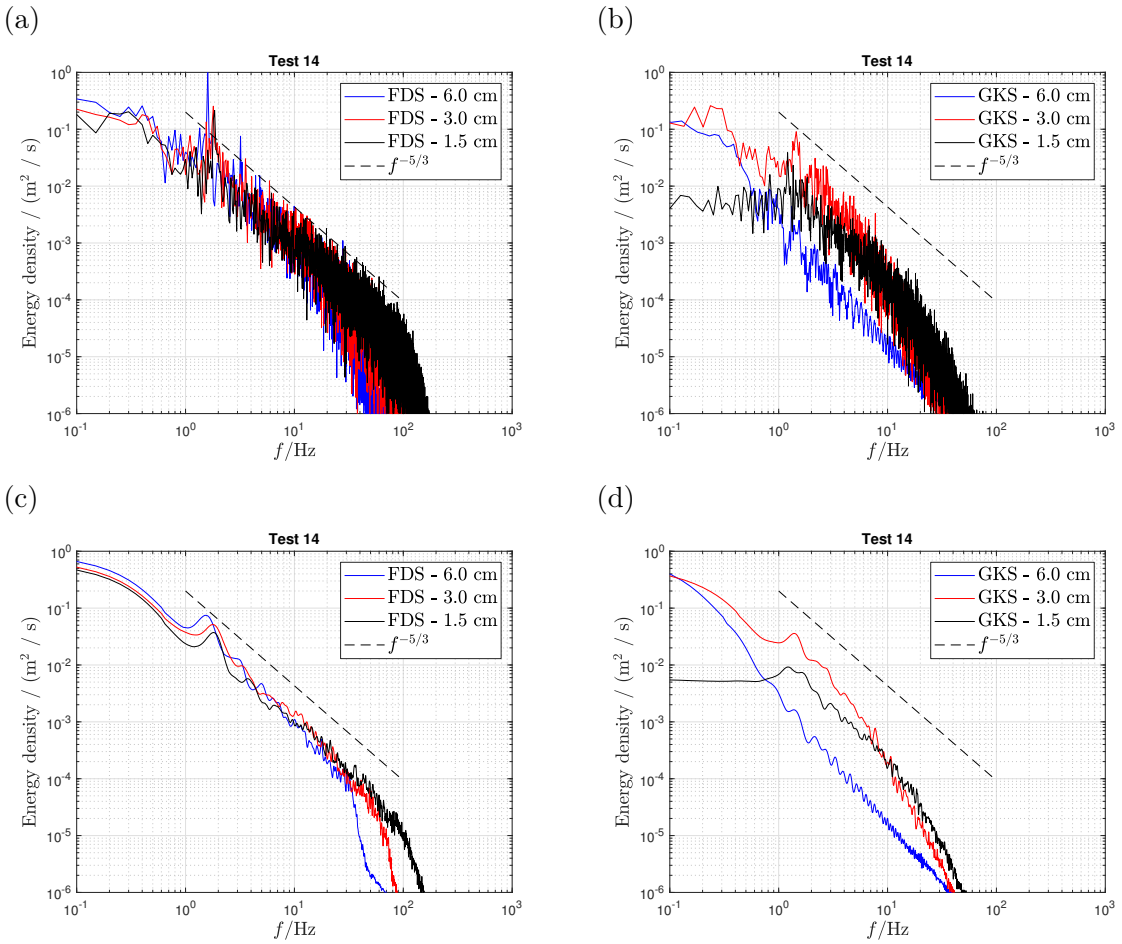


Figure D.8: Sandia flame, Test 14: energy spectra of FDS and VirtualFluidsGKS. (a) and (b) show the original spectra and (c) and (d) the lowpass filtered spectra.

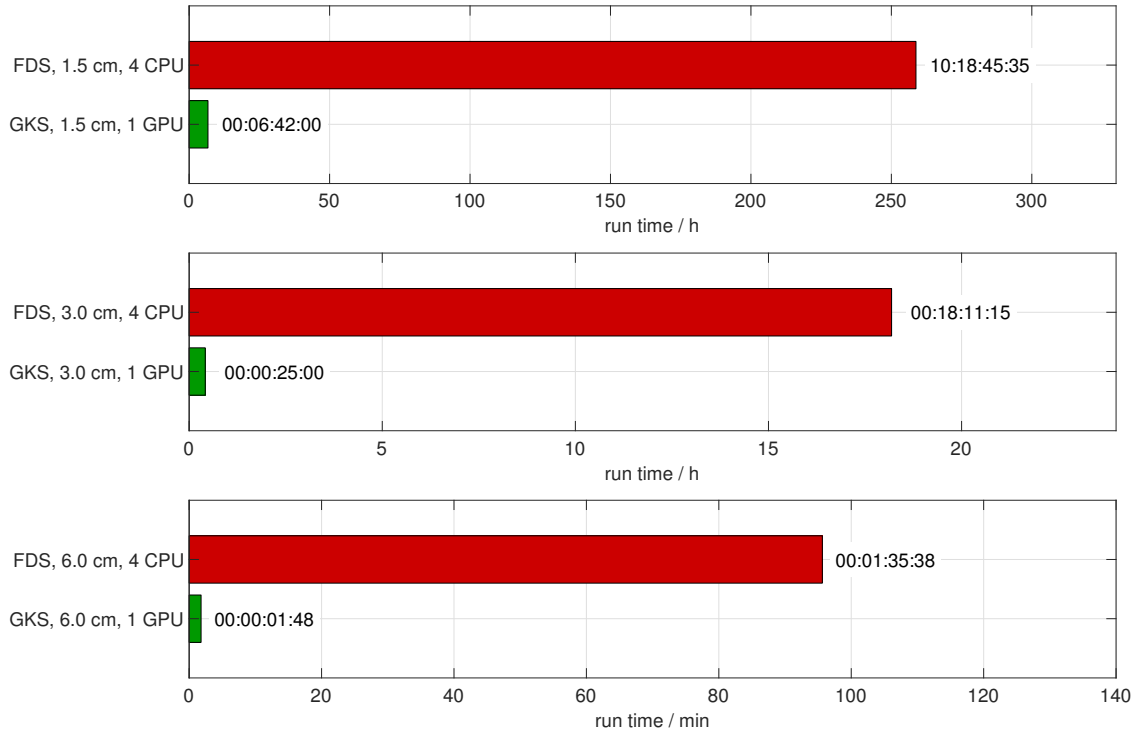


Figure D.9: Sandia flame, Test 14, run times

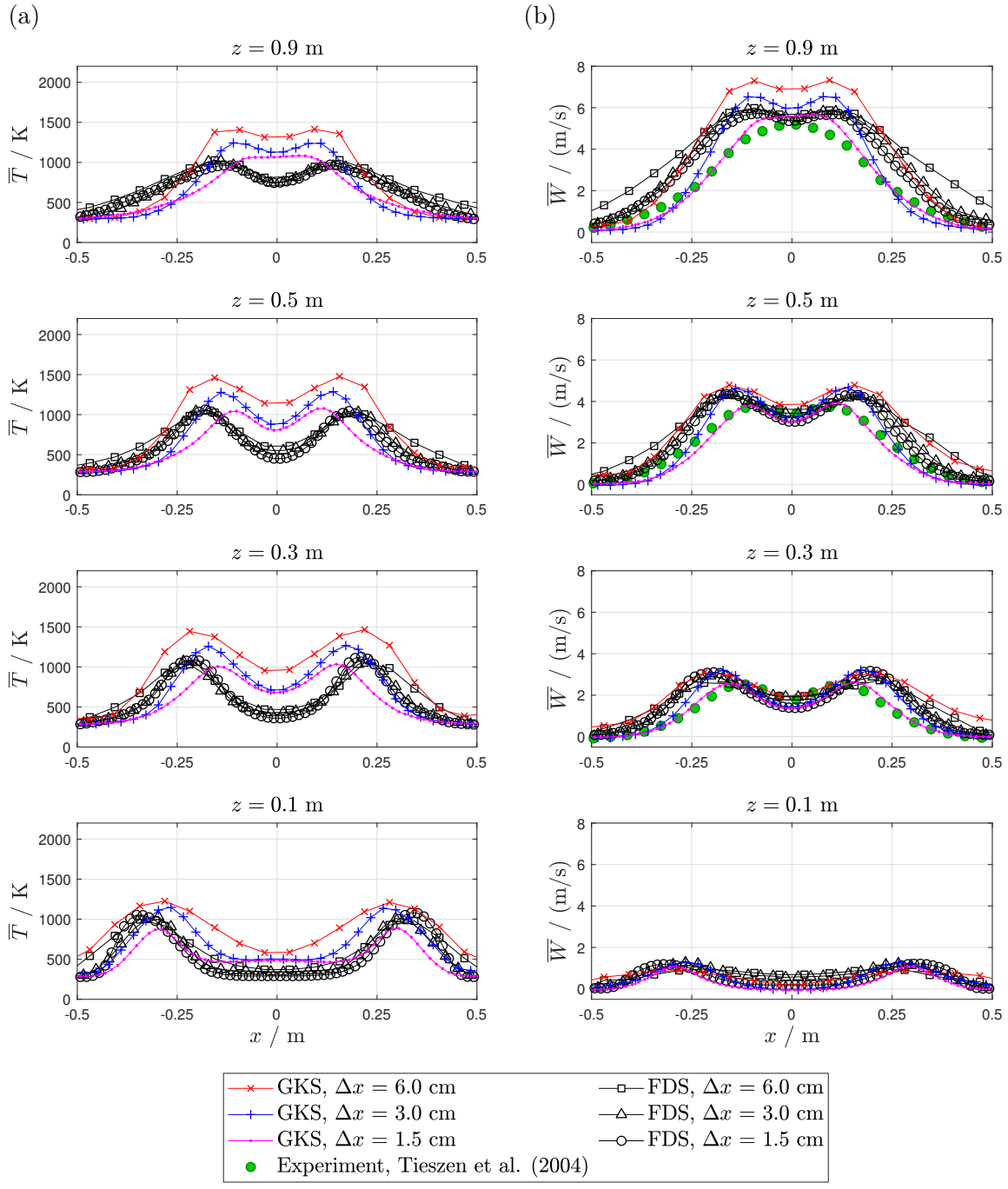


Figure D.10: Sandia flame, Test 14: temperature and vertical velocity profiles

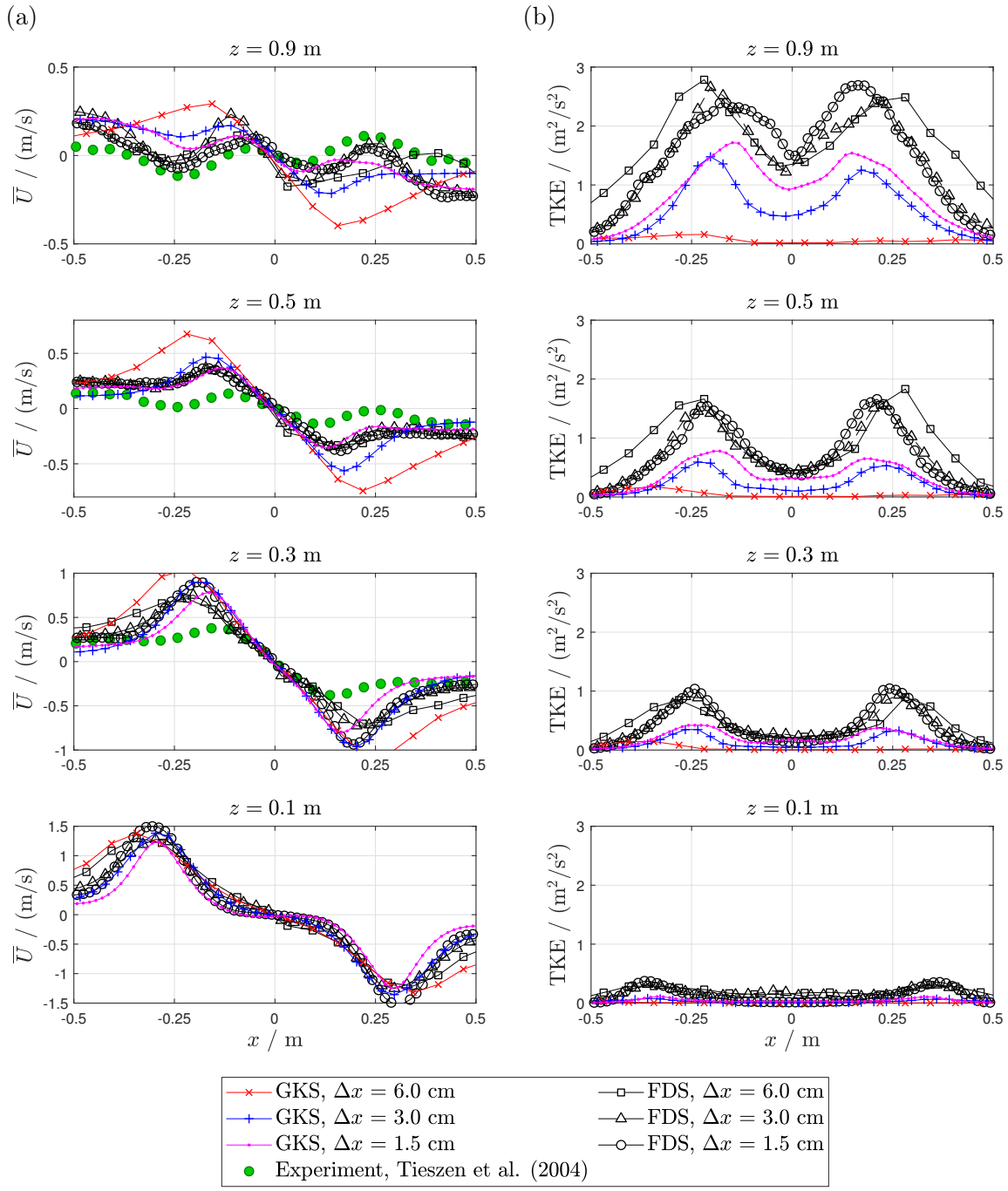


Figure D.11: Sandia flame, Test 14: horizontal velocity and turbulent kinetic energy profiles

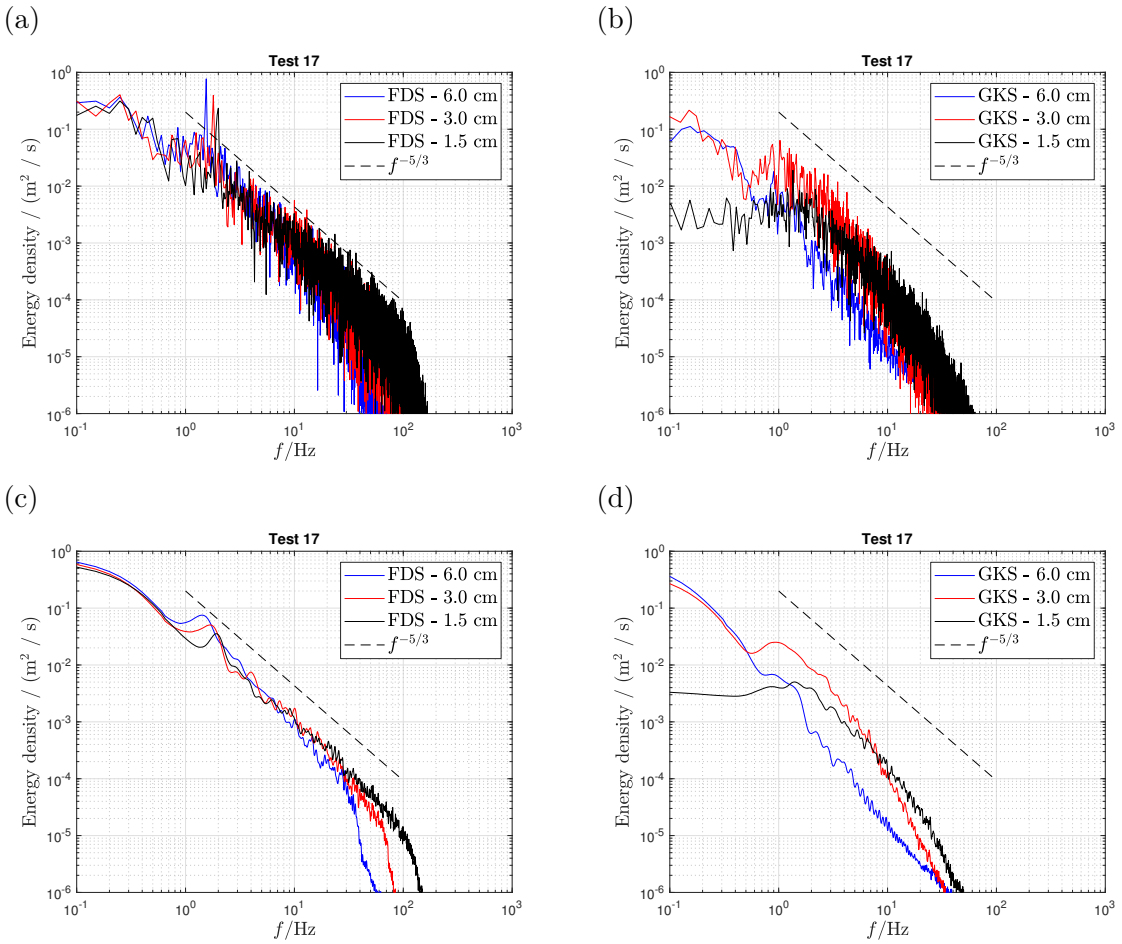


Figure D.12: Sandia flame, Test 17: energy spectra of FDS and VirtualFluidsGKS. (a) and (b) show the original spectra and (c) and (d) the lowpass filtered spectra.

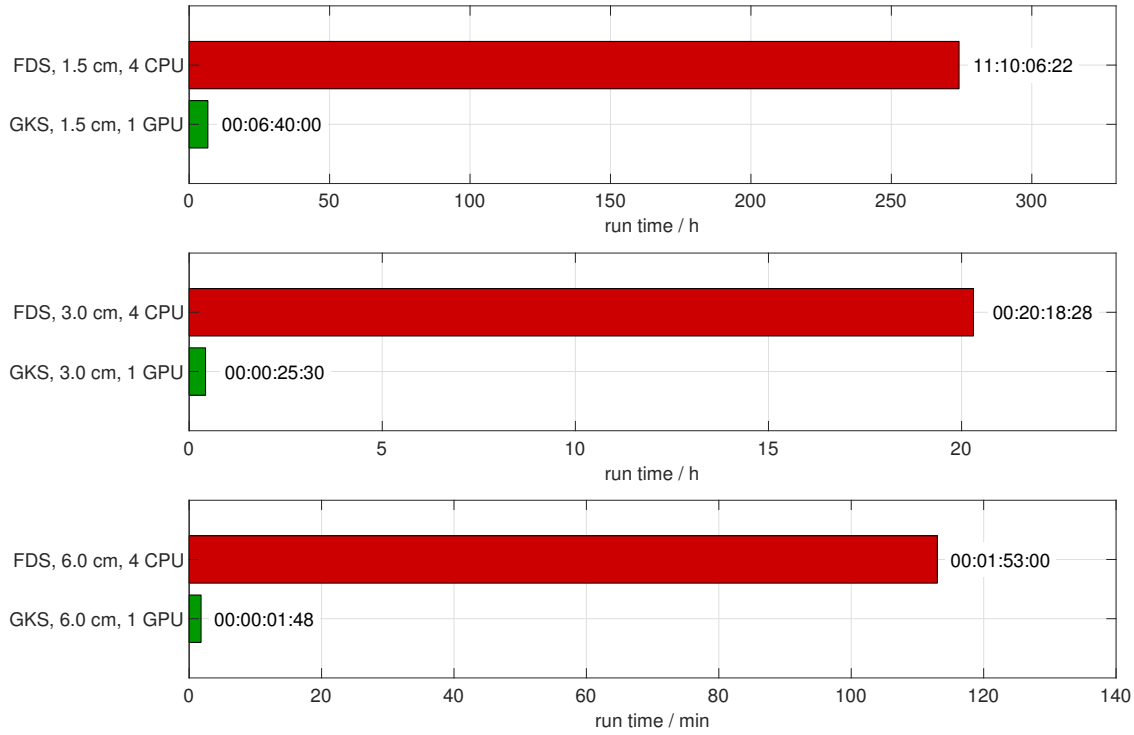


Figure D.13: Sandia flame, Test 17, run times

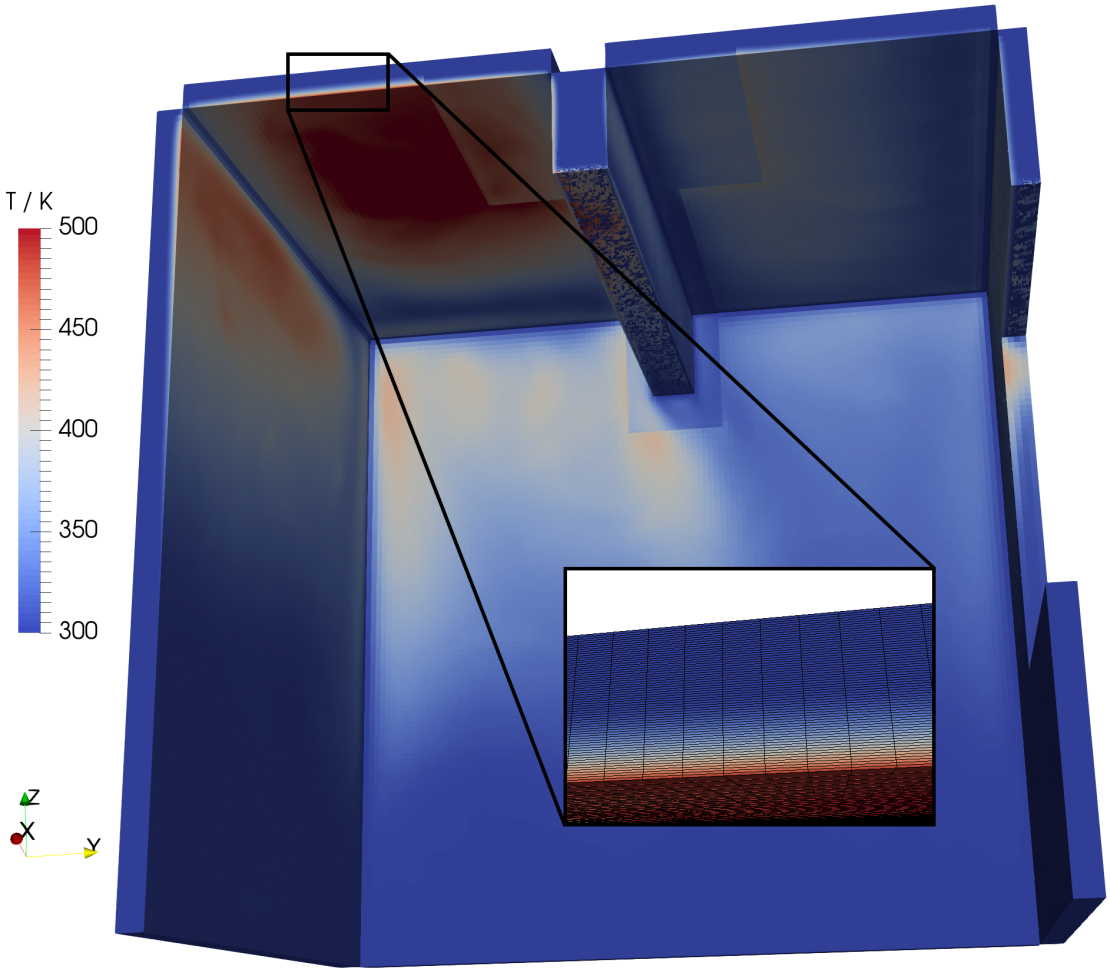


Figure D.14: Compartment Fire: temperatures in the solid

E Detailed Chapman-Enskog expansion

E.1 Euler equations

This appendix shows the detailed derivation of the Navier-Stokes equations based on the Chapman-Enskog expansion. The simpler Euler equations are obtained by computing the conserved moments of the BGK-Boltzmann equation. This yields the Euler equations in the conservation form

$$\begin{pmatrix} \rho \\ \rho U \\ \rho V \\ \rho W \\ \rho E \end{pmatrix}_t + \begin{pmatrix} \rho U \\ \rho U^2 + p \\ \rho UV \\ \rho UW \\ U\left(\rho E + p\right) \end{pmatrix}_x + \begin{pmatrix} \rho V \\ \rho UV \\ \rho V^2 + p \\ \rho VW \\ V\left(\rho E + p\right) \end{pmatrix}_y + \begin{pmatrix} \rho W \\ \rho UW \\ \rho VW \\ \rho W^2 + p \\ W\left(\rho E + p\right) \end{pmatrix}_y = 0. \qquad \text{(E.1)}$$

The Euler equations describe the inviscid fluid flow. Including viscous terms to the right hand side of the Euler equation yields the Navier-Stokes equations. Hence, the left hand side of Euler and Navier-Stokes equations are equal.

In the solution procedure for the right hand side of the Navier-Stokes equations new temporal derivatives appear, that stem from the Chapman-Enskog Expansion. Formally the solution for the Navier-Stokes equation would require solving for the first derivatives in time. This formal solution can be obtained approximately by self insertion of the Navier-Stokes equations. Looking at the shape of the Navier-Stokes equations, one finds, that all terms on the right hand side contain the relaxation time τ, which is assumed to small. Inserting the Navier-Stokes equations in itself (in the time derivatives on the right hand side) yields on the one hand terms with τ that stem from inserting parts of the Euler equation into the right hand side of the Navier-Stokes equations and on the other hand terms with τ square that stem from inserting the Navier-Stokes right hand side. In the derivation of the Chapman-Enskog expansion these τ^2 terms are already neglected. Hence, these terms can also be neglected here, such that insertion of the Euler equation is sufficient. For this purpose the Euler equations are solved for some specific time derivatives.

E.2 Euler time derivatives

Density:

$$\rho_t = -(\rho U)_x - (\rho V)_y - (\rho W)_z \tag{E.2}$$

x-Momentum:

$$(\rho U)_t = -(\rho U^2 + p)_x - (\rho UV)_y - (\rho UW)_z \tag{E.3}$$

Derivative only of velocity:

$$\rho U_t + U\rho_t = -(\rho U^2 + p)_x - (\rho UV)_y - (\rho UW)_z \tag{E.4}$$

$$\rho U_t = -(\rho U^2 + p)_x - (\rho UV)_y - (\rho UW)_z - U\rho_t \tag{E.5}$$

$$\rho U_t = -(\rho U^2 + p)_x - (\rho UV)_y - (\rho UW)_z + U(\rho U)_x + U(\rho V)_y) + U(\rho W)_z) \tag{E.6}$$

$$\begin{aligned}\rho U_t = &- \rho UU_x - U(\rho U)_x - U(\rho V)_y - \rho VU_y - U(\rho W)_z \\ &- \rho WU_z + U(\rho U)_x + U(\rho V)_y + U(\rho W)_z) - p_x\end{aligned} \tag{E.7}$$

$$\rho U_t = -\rho UU_x - \rho VU_y - \rho WU_z - p_x \tag{E.8}$$

Square velocity:

$$(\rho U^2)_t = 2\rho UU_t + U^2\rho_t \tag{E.9}$$

$$(\rho U^2)_t = -2U(\rho UU_x + \rho VU_y + \rho WU_z + p_x) - U^2((\rho U)_x + (\rho V)_y + (\rho W)_z) \tag{E.10}$$

$$(\rho U^2)_t = -2\rho U^2U_x - 2\rho UVU_y - 2\rho UWU_z - 2Up_x - U^2(\rho U)_x - U^2(\rho V)_y - U^2(\rho W)_z \tag{E.11}$$

$$(\rho U^2)_t = -(\rho U^3)_x - (\rho U^2V)_y - (\rho U^2W)_z - 2Up_x \tag{E.12}$$

y-Momentum:

$$(\rho V)_t = -(\rho UV)_x - (\rho V^2 + p)_y - (\rho VW)_z \tag{E.13}$$

$$\rho V_t = -\rho UV_x - \rho VV_y - \rho WV_z - p_y \tag{E.14}$$

$$(\rho V^2)_t = -(\rho UV^2)_x - (\rho V^3)_y - (\rho WV^2)_z - 2Vp_y \tag{E.15}$$

z-Momentum:

$$(\rho W)_t = -(\rho UW)_x - (\rho VW)_y - (\rho W^2 + p)_z \tag{E.16}$$

$$\rho W_t = -\rho UW_x - \rho VW_y - \rho WW_z - p_z \tag{E.17}$$

$$(\rho W^2)_t = -(\rho UW^2)_x - (\rho VW^2)_y - (\rho W^3)_z - 2Wp_z \tag{E.18}$$

Mixed Momentum:

$$(\rho UV)_t = UV\rho_t + \rho VU_t + \rho UV_t \tag{E.19}$$

$$\begin{aligned}(\rho UV)_t = {} & UV(-(\rho U)_x - (\rho V)_y - (\rho W)_z) + + \\ & + V(-\rho UU_x - \rho VU_y - \rho WU_z - p_x) \\ & + U(-\rho UV_x - \rho VV_y - \rho WV_z - p_y)\end{aligned} \tag{E.20}$$

$$\begin{aligned}(\rho UV)_t = {} & (-UV(\rho U)_x - UV(\rho V)_y - UV(\rho W)_z) + + \\ & + (-\rho UVU_x - \rho V^2U_y - \rho VWU_z - Vp_x) \\ & + (-\rho U^2V_x - \rho UVV_y - \rho UWV_z - Up_y)\end{aligned} \tag{E.21}$$

$$(\rho UV)_t = -(\rho U^2V)_x - (\rho UV^2)_y - (\rho UVW)_z - Up_y - Vp_x \tag{E.22}$$

Energy/Pressure:

$$\frac{1}{2}(\rho U^2+\rho V^2+\rho W^2+(K+3)p)_t = -(U(\rho E+p))_x-(V(\rho E+p))_y-(W(\rho E+p))_z \tag{E.23}$$

$$\frac{K+3}{2}p_t = -\frac{1}{2}(\rho U^2 + \rho V^2 + \rho W^2)_t - (U(\rho E + p))_x - (V(\rho E + p))_y - (W(\rho E + p))_z \tag{E.24}$$

Part 1:

$$(U(\rho E + p))_x = (Up)_x + (\rho EU)_x \tag{E.25}$$

$$(U(\rho E+p))_x = (Up)_x+\frac{1}{2}\rho U(U^2+V^2+W^2)_x+\frac{1}{2}(U^2+V^2+W^2)(\rho U)_x+\frac{K+3}{2}(Up)_x \tag{E.26}$$

$$(U(\rho E+p))_x = \frac{K+5}{2}(Up)_x+\frac{1}{2}\rho U(U^2+V^2+W^2)_x+\frac{1}{2}(U^2+V^2+W^2)(\rho U)_x \quad \text{(E.27)}$$

$$(U(\rho E+p))_x = \frac{K+5}{2}(Up)_x+\rho U(UU_x+VV_x+WW_x)+\frac{1}{2}(U^2+V^2+W^2)(\rho U)_x \quad \text{(E.28)}$$

$$(U(\rho E+p))_x = \frac{K+5}{2}(Up)_x+\frac{1}{2}(\rho U^3+\rho UV^2+\rho UW^2)_x \quad \text{(E.29)}$$

And similar:

$$(V(\rho E+p))_y = \frac{K+5}{2}(Vp)_y+\frac{1}{2}(\rho VU^2+\rho V^3+\rho VW^2)_y \quad \text{(E.30)}$$

$$(W(\rho E+p))_z = \frac{K+5}{2}(Wp)_z+\frac{1}{2}(\rho WU^2+\rho WV^2+\rho W^3)_z \quad \text{(E.31)}$$

Finally:

$$\begin{aligned}\frac{K+3}{2}p_t = {} & \tfrac{1}{2}(\rho U^3)_x+\tfrac{1}{2}(\rho U^2V)_y+\tfrac{1}{2}(\rho U^2W)_z+Up_x \\ & +\tfrac{1}{2}(\rho UV^2)_x+\tfrac{1}{2}(\rho V^3)_y+\tfrac{1}{2}(\rho WV^2)_z+Vp_y \\ & +\tfrac{1}{2}(\rho UW^2)_x+\tfrac{1}{2}(\rho VW^2)_y+\tfrac{1}{2}(\rho W^3)_z+Wp_z \\ & -\tfrac{K+5}{2}(Up)_x-\tfrac{1}{2}(\rho U^3+\rho UV^2+\rho UW^2)_x \\ & -\tfrac{K+5}{2}(Vp)_y-\tfrac{1}{2}(\rho VU^2+\rho V^3+\rho VW^2)_y \\ & -\tfrac{K+5}{2}(Wp)_z-\tfrac{1}{2}(\rho WU^2+\rho WV^2+\rho W^3)_z\end{aligned} \quad \text{(E.32)}$$

$$\frac{K+3}{2}p_t = Up_x+Vp_y+Wp_z-\frac{K+5}{2}((Up)_x+(Vp)_y+(Wp)_z) \quad \text{(E.33)}$$

$$\frac{K+3}{2}p_t = Up_x+Vp_y-\frac{K+5}{2}(Up_x+pU_x+Vp_y+pV_y+Wp_z+pW_z) \quad \text{(E.34)}$$

$$\frac{K+3}{2}p_t = -\frac{K+3}{2}(Up_x+Vp_y+Wp_z)-\frac{K+5}{2}(pU_x+pV_y+pW_z) \quad \text{(E.35)}$$

$$p_t = -Up_x-Vp_y-Wp_z-\frac{K+5}{K+3}(pU_x+pV_y+pW_z) \quad \text{(E.36)}$$

E.3 Navier-Stokes equations

Constructing equation:

$$\begin{aligned} RHS = &\int_{\Xi} \underline{\psi}(f_{tt}^{eq} + 2uf_{tx}^{eq} + 2vf_{ty}^{eq} + 2wf_{tz}^{eq})d\Xi \\ &+ \int_{\Xi} \underline{\psi}(u^2 f_{xx}^{eq} + v^2 f_{yy}^{eq} + w^2 f_{zz}^{eq} + 2uvf_{xy}^{eq} + 2uwf_{xz}^{eq} + 2vwf_{yz}^{eq})d\Xi \end{aligned} \tag{E.37}$$

E.3.1 Mass equation:

Part 1:

$$P_1 = \int_{\Xi} (f_{tt}^{eq} + 2uf_{tx}^{eq} + 2vf_{ty}^{eq} + 2wf_{tz}^{eq})d\Xi = \rho_{tt} + 2(\rho U)_{tx} + 2(\rho V)_{ty} + 2(\rho W)_{tz} \tag{E.38}$$

$$P_1 = (\rho_t + 2(\rho U)_x + 2(\rho V)_y + 2(\rho W)_z)_t \tag{E.39}$$

$$P_1 = (-(\rho U)_x - (\rho V)_y - (\rho W)_z + 2(\rho U)_x + 2(\rho V)_y + 2(\rho W)_z)_t \tag{E.40}$$

$$P_1 = (\rho U)_{tx} + (\rho V)_{ty} + (\rho W)_{tz} \tag{E.41}$$

$$\begin{aligned} P_1 = &(-(\rho U^2 + p)_x - (\rho UV)_y - (\rho UW)_z)_x \\ &+ (-(\rho V^2 + p)_y - (\rho UV)_x - (\rho VW)_z)_y \\ &+ (-(\rho W^2 + p)_z - (\rho UW)_x - (\rho VW)_y)_z \end{aligned} \tag{E.42}$$

$$P_1 = -(\rho U^2 + p)_{xx} - (\rho V^2 + p)_{yy} - (\rho W^2 + p)_{yy} - 2(\rho UV)_{xy} - 2(\rho UW)_{xz} - 2(\rho VW)_{yz} \tag{E.43}$$

Part 2:

$$P_2 = \int_{\Xi} (u^2 f_{xx}^{eq} + v^2 f_{yy}^{eq} + w^2 f_{zz}^{eq} + 2uvf_{xy}^{eq} + 2uwf_{xz}^{eq} + 2vwf_{yz}^{eq})d\Xi \tag{E.44}$$

$$P_2 = (\rho U^2 + p)_{xx} + (\rho V^2 + p)_{yy} + (\rho W^2 + p)_{zz} + 2(\rho UV)_{xy} + 2(\rho UW)_{xz} + 2(\rho VW)_{yz} \tag{E.45}$$

Finally:

$$RHS(\rho) = P_1 + P_2 = 0 \tag{E.46}$$

E.3.2 x-Momentum equation

Part 1:

$$P_1 = \int_{\Xi} u(f_{tt}^{eq} + 2uf_{tx}^{eq} + 2vf_{ty}^{eq} + 2vf_{tz}^{eq})d\Xi \tag{E.47}$$

$$P_1 = (\rho U)_{tt} + 2(\rho U^2 + p)_{tx} + 2(\rho UV)_{ty} + 2(\rho UW)_{tz} \tag{E.48}$$

$$P_1 = ((\rho U)_t + 2(\rho U^2 + p)_x + 2(\rho UV)_y + 2(\rho UW)_z)_t \tag{E.49}$$

$$P_1 = ((-(\rho U^2 + p)_x - (\rho UV)_y) - (\rho UW)_z) + 2(\rho U^2 + p)_x + 2(\rho UV)_y + 2(\rho UW)_z)_t)_t \tag{E.50}$$

$$P_1 = ((\rho U^2 + p)_x + (\rho UV)_y + (\rho UW)_z)_t \tag{E.51}$$

$$\begin{aligned} P_1 = &((\rho U^2)_t)_x \\ &+ ((\rho UV)_t)_y \\ &+ ((\rho UW)_t)_z \\ &+ (p_t)_x \end{aligned} \tag{E.52}$$

$$\begin{aligned} P_1 = &(-(\rho U^3)_x - (\rho U^2 V)_y - (\rho U^2 W)_z - 2Up_x)_x \\ &+ (-(\rho U^2 V)_x - (\rho UV^2)_y - (\rho UVW)_z - Up_y - Vp_x)_y \\ &+ (-(\rho U^2 W)_x - (\rho UW^2)_z - (\rho UVW)_y - Up_z - Wp_x)_z \\ &+ (-Up_x - Vp_y - Wp_z - \frac{K+5}{K+3}pU_x - \frac{K+5}{K+3}pV_y - \frac{K+5}{K+3}pW_z)_x \end{aligned} \tag{E.53}$$

$$
\begin{aligned}
P_1 = &(-(\rho U^3)_x - (\rho U^2 V)_y - (\rho U^2 W)_z)_x \\
&+ (-(\rho U^2 V)_x - (\rho U V^2)_y - (\rho U V W)_z)_y \\
&+ (-(\rho U^2 W)_x - (\rho U W^2)_z - (\rho U V W)_y)_z \\
&+ (-3U p_x - V p_y - W p_z - \frac{K+5}{K+3} p U_x - \frac{K+5}{K+3} p V_y - \frac{K+5}{K+3} p W_z)_x \\
&+ (-U p_y - V p_x)_y \\
&+ (-U p_z - W p_x)_z
\end{aligned}
\tag{E.54}
$$

Part 2:

$$
P_2 = \int_{\Xi} u(u^2 f_{xx}^{eq} + v^2 f_{yy}^{eq} + w^2 f_{zz}^{eq} + 2uv f_{xy}^{eq} + 2uw f_{xz}^{eq} + 2vw f_{yz}^{eq}) d\Xi \tag{E.55}
$$

$$
\begin{aligned}
P_2 = &(\rho U^3 + 3Up)_{xx} + (U(\rho V^2 + p))_{yy} + (U(\rho W^2 + p))_{zz} \\
&+ 2(V(\rho U^2 + p))_{xy} + 2(W(\rho U^2 + p))_{xz} + 2(\rho U V W)_{yz}
\end{aligned}
\tag{E.56}
$$

$$
\begin{aligned}
P_2 = &((\rho U^3)_x + (\rho U^2 V)_y + (\rho U^2 W)_z)_x \\
&+ ((\rho U^2 V)_x + (\rho U V^2)_y - (\rho U V W)_z)_y \\
&+ ((\rho U^2 W)_x + (\rho U W^2)_z - (\rho U V W)_y)_z \\
&+ 3(Up)_{xx} + (Up)_{yy} + (Up)_{zz} + 2(Vp)_{xy} + 2(Wp)_{xz}
\end{aligned}
\tag{E.57}
$$

Finally:

$$\begin{aligned} P_1 + P_2 = &(3(Up)_x - 3Up_x - \frac{K+5}{K+3}pU_x)_x \\ &+ (2(Vp)_y - Vp_y - \frac{K+5}{K+3}pV_y)_x \\ &+ (2(Wp)_z - Wp_z - \frac{K+5}{K+3}pW_z)_x \\ &+ ((Up)_y - Up_y - Vp_x)_y \\ &+ ((Up)_z - Up_z - Wp_x)_z \end{aligned} \tag{E.58}$$

$$\begin{aligned} P_1 + P_2 = &(3Up_x + 3pU_x - 3Up_x - \frac{K+5}{K+3}pU_x)_x \\ &+ (2Vp_y + 2pV_y - Vp_y - \frac{K+5}{K+3}pV_y)_x \\ &+ (2Wp_z + 2pW_z - Wp_z - \frac{K+5}{K+3}pW_z)_x \\ &+ (pU_y - Vp_x)_y \\ &+ (pU_z - Wp_x)_z \end{aligned} \tag{E.59}$$

$$\begin{aligned} P_1 + P_2 = &(3pU_x - \frac{K+5}{K+3}pU_x)_x \\ &+ (pV_y - \frac{K+5}{K+3}pV_y + pV_y + Vp_y)_x \\ &+ (pW_z - \frac{K+5}{K+3}pW_z + pW_z + Wp_z)_x \\ &+ (pU_y - (Vp)_x + pV_x)_y \\ &+ (pU_z - (Wp)_x + pW_x)_z \end{aligned} \tag{E.60}$$

$$\begin{aligned} P_1 + P_2 = &(3pU_x - \frac{K+5}{K+3}pU_x)_x \\ &+ (pV_y - \frac{K+5}{K+3}pV_y + (pV)_y)_x \\ &+ (pW_z - \frac{K+5}{K+3}pW_z + (pW)_z)_x \\ &+ (pU_y - (Vp)_x + pV_x)_y \\ &+ (pU_z - (Wp)_x + pW_x)_z \end{aligned} \tag{E.61}$$

$$\begin{aligned} P_1 + P_2 = &(3pU_x - \frac{K+5}{K+3}pU_x)_x \\ &+ (pV_y - \frac{K+5}{K+3}pV_y)_x \\ &+ (pW_z - \frac{K+5}{K+3}pW_z)_x \\ &+ (pU_y + pV_x)_y \\ &+ (pU_z + pW_x)_z \end{aligned} \tag{E.62}$$

$$\begin{aligned} P_1 + P_2 = &((3 - \frac{K+5}{K+3})pU_x)_x \\ &+ ((1 - \frac{K+5}{K+3})pV_y)_x \\ &+ ((1 - \frac{K+5}{K+3})pW_z)_x \\ &+ (pU_y + pV_x)_y \\ &+ (pU_z + pW_x)_z \end{aligned} \tag{E.63}$$

$$P_1 + P_2 = \Big(2pU_x + \big(1 - \frac{K+5}{K+3}\big)p\big(U_x + V_y + W_z\big)\Big)_x + \Big(pU_y + pV_x\Big)_y + \Big(pU_z + pW_x\Big)_z \tag{E.64}$$

In terms of viscosity:

$$RHS(\rho U) = \tau\Big(2pU_x + \big(1 - \frac{K+5}{K+3}\big)p\big(U_x + V_y + W_z\big)\Big)_x + \tau\Big(pU_y + pV_x\Big)_y + \tau\Big(pU_z + pW_x\Big)_z \tag{E.65}$$

The literature reports the right hand side of the momentum equation as [37, pp. 3-21]:

$$RHS(\rho U) = \Big(2\mu U_x + \big(\zeta - \tfrac{2}{3}\mu\big)\big(U_x + V_y + W_x\big)\Big)_x + \Big(\mu U_y + \mu V_x\Big)_y + \Big(\mu U_z + \mu W_x\Big)_z \tag{E.66}$$

Finally, comparison of coefficients yields

$$\mu = \tau p \qquad \text{and} \qquad \zeta = \frac{2}{3}\frac{K}{K+3}\tau p, \tag{E.67}$$

where μ is the dynamic viscosity and ζ is the bulk viscosity. This result is consistent with the findings in [26, Appendix B].

The two other direction can be obtained from symmetry arguments. Hence, the final viscous momentum flux terms are:

$$\begin{aligned}
RHS(\rho U) &= \Big(2\mu U_x + \big(\zeta - \tfrac{2}{3}\mu\big)\big(U_x + V_y + W_x\big)\Big)_x + \Big(\mu U_y + \mu V_x\Big)_y + \Big(\mu U_z + \mu W_x\Big)_z \\
RHS(\rho V) &= \Big(2\mu V_y + \big(\zeta - \tfrac{2}{3}\mu\big)\big(U_x + V_y + W_x\big)\Big)_y + \Big(\mu U_y + \mu V_x\Big)_x + \Big(\mu V_z + \mu W_y\Big)_z \\
RHS(\rho W) &= \Big(2\mu W_z + \big(\zeta - \tfrac{2}{3}\mu\big)\big(U_x + V_y + W_x\big)\Big)_z + \Big(\mu U_z + \mu W_x\Big)_x + \Big(\mu V_z + \mu W_y\Big)_y
\end{aligned} \tag{E.68}$$

E.3.3 Energy equation

Part 1:

$$P_1 = \int_{\Xi} (u^2 + v^2 + w^2 + \xi^2)(f_{tt}^{eq} + 2uf_{tx}^{eq} + 2vf_{ty}^{eq} + 2wf_{tz}^{eq}) d\Xi \tag{E.69}$$

Set in the moments of f^{eq}:

$$\begin{aligned} P_1 = &(\rho U^2 + \rho V^2 + \rho W^2 + (K+3)p)_{tt} \\ &+ 2(\rho U^3 + 3Up + U(\rho V^2 + p + \rho W^2 + p + Kp))_{tx} \\ &+ 2(\rho V^3 + 3Vp + V(\rho U^2 + p + \rho W^2 + p + Kp))_{ty} \\ &+ 2(\rho W^3 + 3Wp + W(\rho U^2 + p + \rho V^2 + p + Kp))_{tz} \end{aligned} \tag{E.70}$$

Expand the latter three lines and collect common terms Up, Vp, Wp:

$$\begin{aligned} P_1 = &(\rho U^2 + \rho V^2 + \rho W^2 + (K+3)p)_{tt} \\ &+ 2(\rho U^3 + \rho UV^2 + \rho UW^2 + (K+5)Up)_{tx} \\ &+ 2(\rho V^3 + \rho U^2 V + \rho VW^2 + (K+5)Vp)_{ty} \\ &+ 2(\rho W^3 + \rho U^2 W + \rho V^2 W + (K+5)Wp)_{tz} \end{aligned} \tag{E.71}$$

Split the first line for better insertion of time derivatives:

$$\begin{aligned} P_1 = &(\rho U^2)_{tt} \\ &+ (\rho V^2)_{tt} \\ &+ (\rho W^2)_{tt} \\ &+ (K+3)(p)_{tt} \\ &+ 2(\rho U^3 + \rho UV^2 + \rho UW^2 + (K+5)Up)_{tx} \\ &+ 2(\rho V^3 + \rho U^2 V + \rho VW^2 + (K+5)Vp)_{ty} \\ &+ 2(\rho W^3 + \rho U^2 W + \rho V^2 W + (K+5)Wp)_{tz} \end{aligned} \tag{E.72}$$

First insertion of time derivatives and move pressure term to the bottom:

$$\begin{aligned}
P_1 = &(-(\rho U^3)_x - (\rho U^2 V)_y - (\rho U^2 W)_z - 2Up_x)_t \\
&+ (-(\rho U V^2)_x - (\rho V^3)_y - (\rho W V^2)_z - 2Vp_y)_t \\
&+ (-(\rho U W^2)_x - (\rho V W^2)_y - (\rho W^3)_z - 2Wp_z)_t \\
&+ 2(\rho U^3 + \rho U V^2 + \rho U W^2 + (K+5)Up)_{tx} \\
&+ 2(\rho V^3 + \rho U^2 V + \rho V W^2 + (K+5)Vp)_{ty} \\
&+ 2(\rho W^3 + \rho U^2 W + \rho V^2 W + (K+5)Wp)_{tz} \\
&+ (K+3)(-Up_x - Vp_y - Wp_z - \frac{K+5}{K+3}(pU_x + pV_y + pW_z))_t
\end{aligned} \tag{E.73}$$

Cancel out various velocity terms that appear once in the time derivative and twice in the remaining moments:

$$\begin{aligned}
P_1 = &(-2Up_x)_t \\
&+ (-2Vp_y)_t \\
&+ (-2Wp_z)_t \\
&+ (\rho U^3 + \rho U V^2 + \rho U W^2 + 2(K+5)Up)_{tx} \\
&+ (\rho V^3 + \rho U^2 V + \rho V W^2 + 2(K+5)Vp)_{ty} \\
&+ (\rho W^3 + \rho U^2 W + \rho V^2 W + 2(K+5)Wp)_{tz} \\
&- (K+3)(Up_x + Vp_y + Wp_z)_t - (K+5)(pU_x + pV_y + pW_z))_t
\end{aligned} \tag{E.74}$$

Combine first three lines with last line and use reverse product rule to obtain $(Up)_{tx}$ and similar:

$$\begin{aligned} P_1 = &(\rho U^3 + \rho UV^2 + \rho UW^2 + 2(K+5)Up)_{tx} \\ &+ (\rho V^3 + \rho U^2 V + \rho VW^2 + 2(K+5)Vp)_{ty} \\ &+ (\rho W^3 + \rho U^2 W + \rho V^2 W + 2(K+5)Wp)_{tz} \\ &- (K+5)((Up)_{tx} + (Vp)_{ty} + (Wp)_{tz}) \end{aligned} \tag{E.75}$$

Cancel out terms:

$$\begin{aligned} P_1 = &(\rho U^3 + \rho UV^2 + \rho UW^2)_{tx} \\ &+ (\rho V^3 + \rho U^2 V + \rho VW^2)_{ty} \\ &+ (\rho W^3 + \rho U^2 W + \rho V^2 W)_{tz} \\ &+ (K+5)((Up)_{tx} + (Vp)_{ty} + (Wp)_{tz}) \end{aligned} \tag{E.76}$$

The second insertion of time integrals is split into four parts. First the pressure part P_{1p}, i.e. the last line, is replaced:

$$\begin{aligned} P_{1p} = &(K+5)(Up)_{tx} \\ &+ (K+5)(Vp)_{ty} \\ &+ (K+5)(Wp)_{tz} \end{aligned} \tag{E.77}$$

Expand time derivatives with product rule for insertion of known time derivatives:

$$\begin{aligned} P_{1p} = &(K+5)(Up_t + \rho U_t \frac{p}{\rho})_x \\ &+ (K+5)(Vp_t + \rho V_t \frac{p}{\rho})_y \\ &+ (K+5)(Wp_t + \rho W_t \frac{p}{\rho})_z \end{aligned} \tag{E.78}$$

Insert time derivatives:

$$\begin{aligned}
P_{1p} = &(K+5)(U(-Up_x - Vp_y - Wp_z - \frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_x \\
&+ (K+5)(V(-Up_x - Vp_y - Wp_z - \frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_y \\
&+ (K+5)(W(-Up_x - Vp_y - Wp_z - \frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_z \\
&+ (K+5)((-\rho UU_x - \rho VU_y - \rho WU_z - p_x)\frac{p}{\rho})_x \\
&+ (K+5)((-\rho UV_x - \rho VV_y - \rho WV_z - p_y)\frac{p}{\rho})_y \\
&+ (K+5)((-\rho UW_x - \rho VW_y - \rho WW_z - p_z)\frac{p}{\rho})_z
\end{aligned} \tag{E.79}$$

Expand parenthesis:

$$\begin{aligned}
P_{1p} = &(K+5)(-U^2p_x - UVp_y - UWp_z - U\frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_x \\
&+ (K+5)(-UVp_x - V^2p_y - VWp_z - V\frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_y \\
&+ (K+5)(-UWp_x - VWp_y - W^2p_z - W\frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_z \\
&+ (K+5)(-UpU_x - VpU_y - WpU_z - \frac{p}{\rho}p_x)_x \\
&+ (K+5)(-UpV_x - VpV_y - WpV_z - \frac{p}{\rho}p_y)_y \\
&+ (K+5)(-UpW_x - VpW_y - WpW_z - \frac{p}{\rho}p_z)_z
\end{aligned} \tag{E.80}$$

Collect inverse product rules for several terms and replace $p/\rho = RT$:

$$\begin{aligned}
P_{1p} = {} & (K+5)(-U(Up)_x - V(Up)_y - W(Up)_z - U\frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_x \\
& + (K+5)(-U(Vp)_x - V(Vp)_y - W(Vp)_z - V\frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_y \\
& + (K+5)(-U(Wp)_x - V(Wp)_y - W(Wp)_z - W\frac{K+5}{K+3}(pU_x + pV_y + pW_z)))_z \\
& + (K+5)(-RTp_x)_x \\
& + (K+5)(-RTp_y)_y \\
& + (K+5)(-RTp_z)_z
\end{aligned} \tag{E.81}$$

Next the three upper lines in P_1 are processed. Due to symmetry in spatial directions this is only done for the first line P_{1x}:

$$P_{1x} = (\rho U^3 + \rho UV^2 + \rho UW^2)_{tx} \tag{E.82}$$

Split for easier insertion of time derivatives:

$$\begin{aligned}
P_{1x} = {} & (\rho U^3)_{tx} \\
& + (\rho UV^2)_{tx} \\
& + (\rho UW^2)_{tx}
\end{aligned} \tag{E.83}$$

Use product rule to obtain known time derivatives:

$$\begin{aligned}
P_{1x} = {} & (U(\rho U^2)_t + \rho U^2 U_t)_x \\
& + (U(\rho V^2)_t + \rho V^2 U_t)_x \\
& + (U(\rho W^2)_t + \rho W^2 U_t)_x
\end{aligned} \tag{E.84}$$

Collect the time derivatives ρU_t in the last line:

$$\begin{aligned} P_{1x} = &(U(\rho U^2)_t)_x \\ &+ (U(\rho V^2)_t)_x \\ &+ (U(\rho W^2)_t)_x \\ &+ ((U^2+V^2+W^2)\rho U_t)_x \end{aligned} \tag{E.85}$$

Insert time derivatives:

$$\begin{aligned} P_{1x} = &(U(-(\rho U^3)_x - (\rho U^2 V)_y - (\rho U^2 W)_z - 2Up_x))_x \\ &+ (U(-(\rho U V^2)_x - (\rho V^3)_y - (\rho W V^2)_z - 2Vp_y))_x \\ &+ (U(-(\rho U W^2)_x - (\rho V W^2)_y - (\rho W^3)_z - 2Wp_z))_x \\ &+ ((U^2+V^2+W^2)(-\rho U U_x - \rho V U_y - \rho W U_z - p_x))_x \end{aligned} \tag{E.86}$$

Expand:

$$\begin{aligned} P_{1x} = &(-U(\rho U^3)_x - U(\rho U^2 V)_y - U(\rho U^2 W)_z - 2U^2 p_x)_x \\ &+ (-U(\rho U V^2)_x - U(\rho V^3)_y - U(\rho W V^2)_z - 2UVp_y)_x \\ &+ (-U(\rho U W^2)_x - U(\rho V W^2)_y - U(\rho W^3)_z - 2UWp_z)_x \\ &+ (-\rho U^3 U_x - \rho U^2 V U_y - \rho U^2 W U_z - U^2 p_x)_x \\ &+ (-\rho U V^2 U_x - \rho V^3 U_y - \rho V^2 W U_z - V^2 p_x)_x \\ &+ (-\rho U W^2 U_x - \rho V W^2 U_y - \rho W^3 U_z - W^2 p_x)_x \end{aligned} \tag{E.87}$$

Collect inverse product rules for several terms:

$$\begin{aligned} P_{1x} = &(-(\rho U^4)_x - (\rho U^3 V)_y - (\rho U^3 W)_z - 3U^2 p_x)_x \\ &+ (-(\rho U^2 V^2)_x - (\rho U V^3)_y - (\rho U W V^2)_z - 2UVp_y - V^2 p_x)_x \\ &+ (-(\rho U^2 W^2)_x - (\rho U V W^2)_y - (\rho U W^3)_z - 2UWp_z - W^2 p_x)_x \end{aligned} \tag{E.88}$$

From symmetry the terms P_{1y} and P_{1z} can be obtained, such that P_1 is complete, where P_{1p} is not written out explicitly:

$$
\begin{aligned}
P_1 = &(-(\rho U^4)_x - (\rho U^3 V)_y - (\rho U^3 W)_z - 3U^2 p_x)_x \\
&+ (-(\rho U^2 V^2)_x - (\rho U V^3)_y - (\rho U V^2 W)_z - 2UV p_y - V^2 p_x)_x \\
&+ (-(\rho U^2 W^2)_x - (\rho U V W^2)_y - (\rho U W^3)_z - 2UW p_z - W^2 p_x)_x \\
&+ (-(\rho U^3 V)_x - (\rho U^2 V^2)_y - (\rho U^2 V W)_z - 2UV p_x - U^2 p_y)_y \\
&+ (-(\rho U V^3)_x - (\rho V^4)_y - (\rho V^3 W)_z - 3V^2 p_y)_y \\
&+ (-(\rho U V W^2)_x - (\rho V^2 W^2)_y - (\rho V W^3)_z - 2VW p_z - W^2 p_y)_y \\
&+ (-(\rho U^3 W)_x - (\rho U^2 V W)_y - (\rho U^2 W^2)_z - 2UW p_x - U^2 p_z)_z \\
&+ (-(\rho U V^2 W)_x - (\rho V^3 W)_y - (\rho V^2 W^2)_z - 2VW p_y - V^2 p_z)_z \\
&+ (-(\rho U W^3)_x - (\rho V W^3)_y - (\rho W^4)_z - 3W^2 p_z)_z \\
&+ P_{1p}
\end{aligned}
\tag{E.89}
$$

Collect mixed terms in lines 4 to 6 and pressure terms in last lines:

$$
\begin{aligned}
P_1 = &-(\rho U^4)_{xx} - (\rho U^2 V^2)_{yy} - (\rho U^2 W^2)_{zz} \\
&- (\rho U^2 V^2)_{xx} - (\rho V^4)_{yy} - (\rho V^2 W^2)_{zz} \\
&- (\rho U^2 W^2)_{xx} - (\rho V^2 W^2)_{yy} - (\rho W^4)_{zz} \\
&- 2(\rho U^3 V)_{xy} - 2(\rho U^3 W)_{xz} - 2(\rho U^2 V W)_{yz} \\
&- 2(\rho U V^3)_{xy} - 2(\rho U V^2 W)_{xz} - 2(\rho V^3 W)_{yz} \\
&- 2(\rho U V W^2)_{xy} - 2(\rho U W^3)_{xz} - 2(\rho V W^3)_{yz} \\
&- (3U^2 p_x + 2UV p_y + 2UW p_z + V^2 p_x + W^2 p_x)_x \\
&- (2UV p_x + 3V^2 p_y + 2VW p_z + U^2 p_y + W^2 p_y)_y \\
&- (2UW p_x + 2VW p_y + 3W^2 p_z + U^2 p_z + V^2 p_z)_z \\
&+ P_{1p}
\end{aligned}
\tag{E.90}
$$

Part 2:

$$P_2 = \int_{\Xi} (u^2+v^2+w^2+\xi^2)(u^2 f_{xx}^{eq}+v^2 f_{yy}^{eq}+w^2 f_{zz}^{eq}+2uv f_{xy}^{eq}+2uw f_{xz}^{eq}+2vw f_{yz}^{eq}) d\Xi \quad \text{(E.91)}$$

$$\begin{aligned}
P_2 = {} & (\rho U^4 + 6pU^2 + 3\frac{p^2}{\rho} + \rho(U^2+\frac{p}{\rho})(V^2+\frac{p}{\rho}) \\
& + \rho(U^2+\frac{p}{\rho})(W^2+\frac{p}{\rho}) + \rho(U^2+p)K\frac{p}{\rho})_{xx} \\
& + (\rho(U^2+\frac{p}{\rho})(V^2+\frac{p}{\rho}) + \rho V^4 + 6pV^2 + 3\frac{p^2}{\rho} \\
& + \rho(V^2+\frac{p}{\rho})(W^2+\frac{p}{\rho}) + \rho(V^2+p)K\frac{p}{\rho})_{yy} \\
& + (\rho(U^2+\frac{p}{\rho})(W^2+\frac{p}{\rho}) + \rho(V^2+\frac{p}{\rho})(W^2+\frac{p}{\rho}) \\
& + \rho W^4 + 6pW^2 + 3\frac{p^2}{\rho} + \rho(W^2+p)K\frac{p}{\rho})_{zz} \\
& + 2(\rho(U^3+3Up)V + \rho(V^3+3Vp)U + \rho(W^2+p)UV + UVKp)_{xy} \\
& + 2(\rho(U^3+3Up)W + \rho(V^2+p)UW + \rho(W^3+3Wp)U + UWKp)_{xz} \\
& + 2(\rho(U^2+p)VW + \rho(V^3+3Vp)W + \rho(W^3+3Wp)V + VWKp)_{yz}
\end{aligned} \quad \text{(E.92)}$$

$$\begin{aligned}
P_2 = {} & (\rho U^4 + 6pU^2 + 3\frac{p^2}{\rho} + \rho U^2V^2 + U^2p + V^2p + \frac{p^2}{\rho} \\
& + \rho U^2W^2 + U^2p + W^2p + \frac{p^2}{\rho} + KU^2p + K\frac{p^2}{\rho})_{xx} \\
& + (\rho U^2V^2 + U^2p + V^2p + \frac{p^2}{\rho} + \rho V^4 + 6pV^2 + 3\frac{p^2}{\rho} \\
& + \rho V^2W^2 + V^2p + W^2p + \frac{p^2}{\rho} + KV^2p + K\frac{p^2}{\rho})_{yy} \\
& + (\rho U^2W^2 + U^2p + W^2p + \frac{p^2}{\rho} + \rho V^2W^2 + V^2p + W^2p + \frac{p^2}{\rho} \\
& + \rho W^4 + 6pW^2 + 3\frac{p^2}{\rho} + KW^2p + K\frac{p^2}{\rho})_{zz} \\
& + 2(\rho U^3V + \rho UV^3 + \rho UVW^2 + (K+7)UVp)_{xy} \\
& + 2(\rho U^3W + \rho UV^2W + \rho UW^3 + (K+7)UWp)_{xz} \\
& + 2(\rho U^2VW + \rho V^3W + \rho VW^3 + (K+7)VWp)_{yz}
\end{aligned} \quad \text{(E.93)}$$

$$
\begin{aligned}
P_2 = &(\rho U^4 + \rho U^2 V^2 + \rho U^2 W^2 + (K+8) U^2 p + V^2 p + W^2 p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ (\rho U^2 V^2 + \rho V^4 + \rho V^2 W^2 + U^2 p + (K+8) V^2 p + W^2 p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ (\rho U^2 W^2 + \rho V^2 W^2 + \rho W^4 + U^2 p + V^2 p + (K+8) W^2 p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ 2(\rho U^3 V + \rho U V^3 + \rho U V W^2 + (K+7) U V p)_{xy} \\
&+ 2(\rho U^3 W + \rho U V^2 W + \rho U W^3 + (K+7) U W p)_{xz} \\
&+ 2(\rho U^2 V W + \rho V^3 W + \rho V W^3 + (K+7) V W p)_{yz}
\end{aligned}
\tag{E.94}
$$

Finally:

$$\begin{aligned}
P_1 = &-(\rho U^4)_{xx} - (\rho U^2 V^2)_{yy} - (\rho U^2 W^2)_{zz} \\
&- (\rho U^2 V^2)_{xx} - (\rho V^4)_{yy} - (\rho V^2 W^2)_{zz} \\
&- (\rho U^2 W^2)_{xx} - (\rho V^2 W^2)_{yy} - (\rho W^4)_{zz} \\
&- 2(\rho U^3 V)_{xy} - 2(\rho U^3 W)_{xz} - 2(\rho U^2 V W)_{yz} \\
&- 2(\rho U V^3)_{xy} - 2(\rho U V^2 W)_{xz} - 2(\rho V^3 W)_{yz} \\
&- 2(\rho U V W^2)_{xy} - 2(\rho U W^3)_{xz} - 2(\rho V W^3)_{yz} \\
&- (3U^2 p_x + 2UV p_y + 2UW p_z + V^2 p_x + W^2 p_x)_x \\
&- (2UV p_x + 3V^2 p_y + 2VW p_z + U^2 p_y + W^2 p_y)_y \\
&- (2UW p_x + 2VW p_y + 3W^2 p_z + U^2 p_z + V^2 p_z)_z \\
&+ P_{1p}
\end{aligned} \tag{E.95}$$

$$\begin{aligned}
P_2 = &(\rho U^4 + \rho U^2 V^2 + \rho U^2 W^2 + (K+8) U^2 p + V^2 p + W^2 p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ (\rho U^2 V^2 + \rho V^4 + \rho V^2 W^2 + U^2 p + (K+8) V^2 p + W^2 p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ (\rho U^2 W^2 + \rho V^2 W^2 + \rho W^4 + U^2 p + V^2 p + (K+8) W^2 p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ 2(\rho U^3 V + \rho U V^3 + \rho U V W^2 + (K+7) U V p)_{xy} \\
&+ 2(\rho U^3 W + \rho U V^2 W + \rho U W^3 + (K+7) U W p)_{xz} \\
&+ 2(\rho U^2 V W + \rho V^3 W + \rho V W^3 + (K+7) V W p)_{yz}
\end{aligned} \tag{E.96}$$

All terms without pressure cancel out:

$$
\begin{aligned}
P_1 + P_2 = &((K+8)U^2 p + V^2 p + W^2 p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ (U^2 p + (K+8)V^2 p + W^2 p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ (U^2 p + V^2 p + (K+8)W^2 p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ 2((K+7)UVp)_{xy} \\
&+ 2((K+7)UWp)_{xz} \\
&+ 2((K+7)VWp)_{yz} \\
&- 2(U^2 p_x + UVp_y + UWp_z)_x \\
&- 2(UVp_x + V^2 p_y + VWp_z)_y \\
&- 2(UWp_x + VWp_y + W^2 p_z)_z \\
&- (V^2 p_x + W^2 p_x)_x \\
&- (U^2 p_y + W^2 p_y)_y \\
&- (U^2 p_z + V^2 p_z)_z \\
&+ P_{1p} - (U^2 p_x)_x - (V^2 p_y)_y - (W^2 p_z)_z
\end{aligned}
\tag{E.97}
$$

Inverse product rule in lines 7-9:

$$
\begin{aligned}
P_1 + P_2 = &((K+8)U^2p + V^2p + W^2p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ (U^2p + (K+8)V^2p + W^2p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ (U^2p + V^2p + (K+8)W^2p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ 2((K+7)UVp)_{xy} \\
&+ 2((K+7)UWp)_{xz} \\
&+ 2((K+7)VWp)_{yz} \\
&- 2((U^2p)_x - 2UpU_x + (UVp)_y - UpV_y - VpU_y \\
&+ (UWp)_z - UpW_z - WpU_z)_x \\
&- 2((UVp)_x - UpV_x - VpU_x + (V^2p)_y - 2VpV_y \\
&+ (VWp)_z - VpW_z - WpV_z)_y \\
&- 2((UWp)_x - UpW_x - WpU_x + (VWp)_y - VpW_y - WpV_yV \\
&+ (W^2p)_z - 2WpW_z)_z \\
&- (V^2p_x + W^2p_x)_x \\
&- (U^2p_y + W^2p_y)_y \\
&- (U^2p_z + V^2p_z)_z \\
&+ P_{1p} - (U^2p_x)_x - (V^2p_y)_y - (W^2p_z)_z
\end{aligned}
\tag{E.98}
$$

Cancel out terms $(U^2p)_{xx}$ and $(UVp)_{xy}$ and similar:

$$\begin{aligned}
P_1 + P_2 = &((K+6)U^2p + V^2p + W^2p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ (U^2p + (K+6)V^2p + W^2p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ (U^2p + V^2p + (K+6)W^2p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ 2((K+5)UVp)_{xy} \\
&+ 2((K+5)UWp)_{xz} \\
&+ 2((K+5)VWp)_{yz} \\
&+ 2(2UpU_x + UpV_y + VpU_y + UpW_z + WpU_z)_x \\
&+ 2(UpV_x + VpU_x + 2VpV_y + VpW_z + WpV_z)_y \\
&+ 2(UpW_x + WpU_x + VpW_y + WpV_yV + 2WpW_z)_z \\
&- (V^2p_x + W^2p_x)_x \\
&- (U^2p_y + W^2p_y)_y \\
&- (U^2p_z + V^2p_z)_z \\
&+ P_{1p} - (U^2p_x)_x - (V^2p_y)_y - (W^2p_z)_z
\end{aligned} \tag{E.99}$$

Expand terms $(V^2p)_{xx}$ in per product rule: Also split the mixed derivative terms in two parts with exchanged differentiation order, where the derivatives $(UVp)_y$ and similar are expend per product rule:

Cancel out terms $(V^2p_x)_x$ and similar:

$$
\begin{aligned}
P_1 + P_2 = &((K+6)U^2p + (K+5)\frac{p^2}{\rho})_{xx} + (V^2p_x + W^2p_x)_x \\
&+ ((K+6)V^2p + (K+5)\frac{p^2}{\rho})_{yy} + (U^2p_y + W^2p_y)_y \\
&+ ((K+6)W^2p + (K+5)\frac{p^2}{\rho})_{zz} + (U^2p_z + V^2p_z)_z \\
&+ (2VpV_x + 2WpW_x)_x \\
&+ (2UpU_y + 2WpW_y)_y \\
&+ (2UpU_z + 2VpV_z)_z \\
&+ (K+5)(UVp_y + UpV_y + VpU_y)_x + (K+5)(UVp_x + UpV_x + VpU_x)_y \\
&+ (K+5)(UWp_x + UpW_x + WpU_x)_z + (K+5)(UWp_z + UpW_z + WpU_z)_x \\
&+ (K+5)(VWp_z + VpW_z + WpV_z)_y + (K+5)(VWp_y + VpW_y + WpV_y)_z \\
&+ 2(2UpU_x + UpV_y + VpU_y + UpW_z + WpU_z)_x \\
&+ 2(UpV_x + VpU_x + 2VpV_y + VpW_z + WpV_z)_y \\
&+ 2(UpW_x + WpU_x + VpW_y + WpV_y + 2WpW_z)_z \\
&- (V^2p_x + W^2p_x)_x \\
&- (U^2p_y + W^2p_y)_y \\
&- (U^2p_z + V^2p_z)_z \\
&+ P_{1p} - (U^2p_x)_x - (V^2p_y)_y - (W^2p_z)_z
\end{aligned}
\tag{E.100}
$$

Cancel out terms $(V^2 p_x)_x$ and similar. Also resort the mixed derivative terms:

$$\begin{aligned}
P_1 + P_2 = &((K+6)U^2 p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ ((K+6)V^2 p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ ((K+6)W^2 p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ 2(VpV_x + WpW_x + VpU_y + WpU_z)_x \\
&+ 2(UpU_y + WpW_y + UpV_x + WpV_z)_y \\
&+ 2(UpU_z + VpV_z + UpW_x + VpW_y)_z \\
&+ (K+5)(UVp_y + UpV_y + VpU_y + UWp_z + UpW_z + WpU_z)_x \\
&+ (K+5)(VWp_z + VpW_z + WpV_z + UVp_x + UpV_x + VpU_x)_y \\
&+ (K+5)(UWp_x + UpW_x + WpU_x + VWp_y + VpW_y + WpV_y)_z \\
&+ 2(2UpU_x + UpV_y + UpW_z)_x \\
&+ 2(2VpV_y + VpU_x + VpW_z)_y \\
&+ 2(2WpW_z + WpU_x + WpV_y)_z \\
&+ P_{1p} - (U^2 p_x)_x - (V^2 p_y)_y - (W^2 p_z)_z
\end{aligned} \tag{E.101}$$

Resort the mixed derivatives and expand a single instance of $(U^2p)_{xx}$ and similar:

$$\begin{aligned}
P_1 + P_2 = &((K+5)U^2p + (K+5)\frac{p^2}{\rho})_{xx} + (2UpU_x + U^2p_x)_x \\
&+ ((K+5)V^2p + (K+5)\frac{p^2}{\rho})_{yy} + (2VpV_y + U^2p_y)_y \\
&+ ((K+5)W^2p + (K+5)\frac{p^2}{\rho})_{zz} + (2WpW_z + U^2p_z)_x \\
&+ 2(Vp(V_x + U_y) + Wp(W_x + U_z))_x \\
&+ 2(Up(U_y + V_x) + Wp(W_y + V_z))_y \\
&+ 2(Up(U_z + W_x) + Vp(V_z + W_y))_z \\
&+ (K+5)(UpV_y + VpU_y + UpW_z + WpU_z)_x + (K+5)(UVp_y + UWp_z)_x \\
&+ (K+5)(VpW_z + WpV_z + UpV_x + VpU_x)_y + (K+5)(VWp_z + UVp_x)_y \\
&+ (K+5)(UpW_x + WpU_x + VpW_y + WpV_y)_z + (K+5)(UWp_x + VWp_y)_z \\
&+ 2(2UpU_x + UpV_y + UpW_z)_x \\
&+ 2(2VpV_y + VpU_x + VpW_z)_y \\
&+ 2(2WpW_z + WpU_x + WpV_y)_z \\
&+ P_{1p} - (U^2p_x)_x - (V^2p_y)_y - (W^2p_z)_z
\end{aligned} \tag{E.102}$$

Resort the mixed derivatives to form the new lines 7 - 9 and cancel out terms $(U^2 p_x)_x$ and similar:

$$
\begin{aligned}
P_1 + P_2 = &((K+5)U^2 p + (K+5)\frac{p^2}{\rho})_{xx} \\
&+ ((K+5)V^2 p + (K+5)\frac{p^2}{\rho})_{yy} \\
&+ ((K+5)W^2 p + (K+5)\frac{p^2}{\rho})_{zz} \\
&+ (K+5)(UpV_y + VpU_y + UpW_z + WpU_z)_x + (K+5)(UVp_y + UWp_z)_x \\
&+ (K+5)(VpW_z + WpV_z + UpV_x + VpU_x)_y + (K+5)(VWp_z + UVp_x)_y \\
&+ (K+5)(UpW_x + WpU_x + VpW_y + WpV_y)_z + (K+5)(UWp_x + VWp_y)_z \\
&+ 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + Up(U_x + V_y + W_z)\big)_x \\
&+ 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + Vp(V_y + U_x + W_z)\big)_y \\
&+ 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + Wp(W_z + U_x + V_y)\big)_z \\
&+ P_{1p}
\end{aligned}
\tag{E.103}
$$

Expand the terms $(U^2 p_x)_x$ in the first three lines:

$$
\begin{aligned}
P_1 + P_2 = &((K+5)\frac{p^2}{\rho})_{xx} + (K+5)(2UpU_x + U^2 p_x)_x \\
&+ ((K+5)\frac{p^2}{\rho})_{yy} + (K+5)(2VpV_y + V^2 p_y)_y \\
&+ ((K+5)\frac{p^2}{\rho})_{zz} + (K+5)(2WpW_z + W^2 p_z)_z \\
&+ (K+5)(Up(V_y + W_z) + VpU_y + WpU_z)_x + (K+5)(UVp_y + UWp_z)_x \\
&+ (K+5)(Vp(W_z + U_x) + WpV_z + UpV_x)_y + (K+5)(VWp_z + UVp_x)_y \\
&+ (K+5)(Wp(U_x + V_y) + UpW_x + VpW_y)_z + (K+5)(UWp_x + VWp_y)_z \\
&+ 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + Up(U_x + V_y + W_z)\big)_x \\
&+ 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + Vp(V_y + U_x + W_z)\big)_y \\
&+ 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + Wp(W_z + U_x + V_y)\big)_z \\
&+ P_{1p}
\end{aligned}
\tag{E.104}
$$

Split the expanded terms up:

$$
\begin{aligned}
P_1 + P_2 = {} & ((K+5)\frac{p^2}{\rho})_{xx} + ((K+5)U^2 p_x)_x + (K+5)(UpU_x)_x + (K+5)(UpU_x)_x \\
& + ((K+5)\frac{p^2}{\rho})_{yy} + ((K+5)V^2 p_y)_y + (K+5)(VpV_y)_y + (K+5)(VpV_y)_y \\
& + ((K+5)\frac{p^2}{\rho})_{zz} + ((K+5)W^2 p_z)_z + (K+5)(WpW_z)_z + (K+5)(WpW_z)_z \\
& + (K+5)(Up(V_y + W_z) + VpU_y + WpU_z)_x + (K+5)(UVp_y + UWp_z)_x \\
& + (K+5)(Vp(W_z + U_x) + WpV_z + UpV_x)_y + (K+5)(VWp_z + UVp_x)_y \\
& + (K+5)(Wp(U_x + V_y) + UpW_x + VpW_y)_z + (K+5)(UWp_x + VWp_y)_z \\
& + 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + Up(U_x + V_y + W_z)\big)_x \\
& + 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + Vp(V_y + U_x + W_z)\big)_y \\
& + 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + Wp(W_z + U_x + V_y)\big)_z \\
& + P_{1p}
\end{aligned}
\tag{E.105}
$$

One of the split terms is combined with the first terms of line 4-6 and added to lines 7-9:

$$
\begin{aligned}
P_1 + P_2 = &((K+5)\frac{p^2}{\rho})_{xx} + ((K+5)U^2 p_x)_x + (K+5)(UpU_x)_x \\
&+ ((K+5)\frac{p^2}{\rho})_{yy} + ((K+5)V^2 p_y)_y + (K+5)(VpV_y)_y \\
&+ ((K+5)\frac{p^2}{\rho})_{zz} + ((K+5)W^2 p_z)_z + (K+5)(WpW_z)_z \\
&+ (K+5)(UVp_y + UWp_z + VpU_y + WpU_z)_x \\
&+ (K+5)(VWp_z + UVp_x + WpV_z + UpV_x)_y \\
&+ (K+5)(UWp_x + VWp_y + UpW_x + VpW_y)_z \\
&+ 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + \frac{K+7}{2}Up(U_x + V_y + W_z)\big)_x \\
&+ 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + \frac{K+7}{2}Vp(V_y + U_x + W_z)\big)_y \\
&+ 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + \frac{K+7}{2}Wp(W_z + U_x + V_y)\big)_z \\
&+ P_{1p}
\end{aligned}
\tag{E.106}
$$

Replace p/ρ by RT:

$$
\begin{aligned}
P_1 + P_2 = {} & (K+5)(pRT)_{xx} + ((K+5)U^2 p_x)_x + (K+5)(UpU_x)_x \\
& + (K+5)(pRT)_{yy} + ((K+5)V^2 p_y)_y + (K+5)(VpV_y)_y \\
& + (K+5)(pRT)_{zz} + ((K+5)W^2 p_z)_z + (K+5)(WpW_z)_z \\
& + (K+5)(UVp_y + UWp_z + VpU_y + WpU_z)_x \\
& + (K+5)(VWp_z + UVp_x + WpV_z + UpV_x)_y \\
& + (K+5)(UWp_x + VWp_y + UpW_x + VpW_y)_z \\
& + 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + \frac{K+7}{2} Up(U_x + V_y + W_z)\big)_x \\
& + 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + \frac{K+7}{2} Vp(V_y + U_x + W_z)\big)_y \\
& + 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + \frac{K+7}{2} Wp(W_z + U_x + V_y)\big)_z \\
& + P_{1p}
\end{aligned}
\tag{E.107}
$$

Expand the terms $(pRT)_{xx}$ and similar:

$$
\begin{aligned}
P_1 + P_2 = {} & (K+5)(pRT_x + RTp_x)_x + ((K+5)U^2 p_x)_x + (K+5)(UpU_x)_x \\
& + (K+5)(pRT_y + RTp_y)_y + ((K+5)V^2 p_y)_y + (K+5)(VpV_y)_y \\
& + (K+5)(pRT_z + RTp_z)_z + ((K+5)W^2 p_z)_z + (K+5)(WpW_z)_z \\
& + (K+5)(UVp_y + UWp_z + VpU_y + WpU_z)_x \\
& + (K+5)(VWp_z + UVp_x + WpV_z + UpV_x)_y \\
& + (K+5)(UWp_x + VWp_y + UpW_x + VpW_y)_z \\
& + 2\Big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + \frac{K+7}{2} Up(U_x + V_y + W_z)\Big)_x \\
& + 2\Big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + \frac{K+7}{2} Vp(V_y + U_x + W_z)\Big)_y \\
& + 2\Big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + \frac{K+7}{2} Wp(W_z + U_x + V_y)\Big)_z \\
& + P_{1p}
\end{aligned}
\tag{E.108}
$$

Put the heat conduction term at the correct location:

$$
\begin{aligned}
P_1 + P_2 = {} & (K+5)(RTp_x)_x + ((K+5)U^2p_x)_x + (K+5)(UpU_x)_x \\
& + (K+5)(RTp_y)_y + ((K+5)V^2p_y)_y + (K+5)(VpV_y)_y \\
& + (K+5)(RTp_z)_z + ((K+5)W^2p_z)_z + (K+5)(WpW_z)_z \\
& + (K+5)(UVp_y + UWp_z + VpU_y + WpU_z)_x \\
& + (K+5)(VWp_z + UVp_x + WpV_z + UpV_x)_y \\
& + (K+5)(UWp_x + VWp_y + UpW_x + VpW_y)_z \\
& + 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + p\frac{K+5}{2}RT_x \\
& + \frac{K+7}{2}Up(U_x + V_y + W_z)\big)_x \\
& + 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + p\frac{K+5}{2}RT_y \\
& + \frac{K+7}{2}Vp(V_y + U_x + W_z)\big)_y \\
& + 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + p\frac{K+5}{2}RT_z \\
& + \frac{K+7}{2}Wp(W_z + U_x + V_y)\big)_z \\
& + P_{1p}
\end{aligned}
\tag{E.109}
$$

Insert P_{1p}:

$$
\begin{aligned}
P_1 + P_2 = {} & (K+5)(RTp_x)_x + ((K+5)U^2p_x)_x + (K+5)(UpU_x)_x \\
& + (K+5)(RTp_y)_y + ((K+5)V^2p_y)_y + (K+5)(VpV_y)_y \\
& + (K+5)(RTp_z)_z + ((K+5)W^2p_z)_z + (K+5)(WpW_z)_z \\
& + (K+5)(UVp_y + UWp_z + VpU_y + WpU_z)_x \\
& + (K+5)(VWp_z + UVp_x + WpV_z + UpV_x)_y \\
& + (K+5)(UWp_x + VWp_y + UpW_x + VpW_y)_z \\
& + 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + p\frac{K+5}{2}RT_x \\
& + \frac{K+7}{2}Up(U_x + V_y + W_z)\big)_x \\
& + 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + p\frac{K+5}{2}RT_y \\
& + \frac{K+7}{2}Vp(V_y + U_x + W_z)\big)_y \\
& + 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + p\frac{K+5}{2}RT_z \\
& + \frac{K+7}{2}Wp(W_z + U_x + V_y)\big)_z \\
& + (K+5)(-U(Up)_x - V(Up)_y - W(Up)_z \\
& - U\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_x \\
& + (K+5)(-U(Vp)_x - V(Vp)_y - W(Vp)_z \\
& - V\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_y \\
& + (K+5)(-U(Wp)_x - V(Wp)_y - W(Wp)_z \\
& - W\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_z \\
& + (K+5)(-RTp_x)_x + (K+5)(-RTp_y)_y + (K+5)(-RTp_z)_z
\end{aligned}
\tag{E.110}
$$

Cancel out terms and collect terms in line 4-6 by inverse product rule:

$$
\begin{aligned}
P_1 + P_2 = &((K+5)U^2 p_x)_x + (K+5)(UpU_x)_x \\
&+ ((K+5)V^2 p_y)_y + (K+5)(VpV_y)_y \\
&+ ((K+5)W^2 p_z)_z + (K+5)(WpW_z)_z \\
&+ (K+5)(V(Up)_y + W(Up)_z)_x \\
&+ (K+5)(W(Vp)_z + U(Vp)_x)_y \\
&+ (K+5)(U(Wp)_x + V(Wp)_y)_z \\
&+ 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + p\frac{K+5}{2}RT_x \\
&+ \frac{K+7}{2}Up(U_x + V_y + W_z)\big)_x \\
&+ 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + p\frac{K+5}{2}RT_y \\
&+ \frac{K+7}{2}Vp(V_y + U_x + W_z)\big)_y \\
&+ 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + p\frac{K+5}{2}RT_z \\
&+ \frac{K+7}{2}Wp(W_z + U_x + V_y)\big)_z \\
&+ (K+5)(-U(Up)_x - V(Up)_y - W(Up)_z \\
&- U\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_x \\
&+ (K+5)(-U(Vp)_x - V(Vp)_y - W(Vp)_z \\
&- V\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_y \\
&+ (K+5)(-U(Wp)_x - V(Wp)_y - W(Wp)_z \\
&- W\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_z
\end{aligned}
\tag{E.111}
$$

Cancels out various terms and expand derivative in lines 1-3:

$$
\begin{aligned}
P_1 + P_2 = {} & ((K+5)U^2 p_x)_x + (K+5)(U(pU)_x - U^2 p_x)_x \\
& + ((K+5)V^2 p_y)_y + (K+5)(V(pV)_y - V^2 p_y)_y \\
& + ((K+5)W^2 p_z)_z + (K+5)(W(pW)_z - W^2 p_z)_z \\
& + 2\big(2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + p\frac{K+5}{2}RT_x \\
& + \frac{K+7}{2}Up(U_x + V_y + W_z)\big)_x \\
& + 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + p\frac{K+5}{2}RT_y \\
& + \frac{K+7}{2}Vp(V_y + U_x + W_z)\big)_y \\
& + 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + p\frac{K+5}{2}RT_z \\
& + \frac{K+7}{2}Wp(W_z + U_x + V_y)\big)_z \\
& + (K+5)(-U(Up)_x - U\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_x \\
& + (K+5)(-V(Vp)_y - V\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_y \\
& + (K+5)(-W(Wp)_z - W\frac{K+5}{K+3}(pU_x + pV_y + pW_z))_z
\end{aligned}
\tag{E.112}
$$

Cancel out terms:

$$
\begin{aligned}
P_1 + P_2 = 2\big(&2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + p\frac{K+5}{2}RT_x \\
&+ \frac{K+7}{2}Up(U_x + V_y + W_z)\big)_x \\
&+ 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + p\frac{K+5}{2}RT_y \\
&+ \frac{K+7}{2}Vp(V_y + U_x + W_z)\big)_y \\
&+ 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + p\frac{K+5}{2}RT_z \\
&+ \frac{K+7}{2}Wp(W_z + U_x + V_y)\big)_z \\
&+ (K+5)(-\frac{K+5}{K+3}Up(U_x + V_y + W_z))_x \\
&+ (K+5)(-\frac{K+5}{K+3}Vp(U_x + V_y + W_z))_y \\
&+ (K+5)(-\frac{K+5}{K+3}Wp(U_x + V_y + W_z))_z
\end{aligned}
\tag{E.113}
$$

$$
\begin{aligned}
P_1 + P_2 = 2\big(&2UpU_x + Vp(V_x + U_y) + Wp(W_x + U_z) + p\frac{K+5}{2}RT_x\big)_x \\
&+ 2\big(2VpV_y + Up(U_y + V_x) + Wp(W_y + V_z) + p\frac{K+5}{2}RT_y\big)_y \\
&+ 2\big(2WpW_z + Up(U_z + W_x) + Vp(V_z + W_y) + p\frac{K+5}{2}RT_z\big)_z \\
&- \Big((K+7) - (K+5)\frac{K+5}{K+3}\Big)(Up(U_x + V_y + W_z))_x \\
&- \Big((K+7) - (K+5)\frac{K+5}{K+3}\Big)(Vp(U_x + V_y + W_z))_y \\
&- \Big((K+7) - (K+5)\frac{K+5}{K+3}\Big)(Wp(U_x + V_y + W_z))_z
\end{aligned}
\tag{E.114}
$$

Intermediate simplification

$$\begin{aligned}
(K+7)-(K+5)\frac{K+5}{K+3} &= 2+(K+5)-(K+5)\frac{K+5}{K+3} \\
&= 2+(K+5)\left(1-\frac{K+5}{K+3}\right) \\
&= 2+(K+5)\left(\frac{K+3}{K+3}-\frac{K+5}{K+3}\right) \\
&= 2+(K+5)\left(\frac{K+3-(K+5)}{K+3}\right) \\
&= 2+(K+5)\left(\frac{-2}{K+3}\right) \\
&= 2-2\frac{K+5}{K+3} \\
&= 2(1-\frac{K+5}{K+3})
\end{aligned} \tag{E.115}$$

$$\begin{aligned}
P_1+P_2 = 2\big(&2UpU_x+Vp(V_x+U_y)+Wp(W_x+U_z)+p\frac{K+5}{2}RT_x \\
&+(1-\frac{K+5}{K+3})Up(U_x+V_y+W_z)\big)_x \\
&+2\big(2VpV_y+Up(U_y+V_x)+Wp(W_y+V_z)+p\frac{K+5}{2}RT_y \\
&+(1-\frac{K+5}{K+3})Vp(U_x+V_y+W_z)\big)_y \\
&+2\big(2WpW_z+Up(U_z+W_x)+Vp(V_z+W_y)+p\frac{K+5}{2}RT_z \\
&+(1-\frac{K+5}{K+3})Wp(U_x+V_y+W_z)\big)_z
\end{aligned} \tag{E.116}$$

The energy equation is recovered as:

$$\begin{aligned} RHS(\rho E) = &\big(2\mu U U_x \;\; + \mu V(V_x + U_y) \;\; + \mu W(W_x + U_z) + \kappa T_x + (\zeta - \tfrac{2}{3}\mu) U(U_x + V_y + W_z)\big)_x \\ &+ \big(2\mu V V_y \;\; + \mu U(U_y + V_x) \;\; + \mu W(W_y + V_z) + \kappa T_y + (\zeta - \tfrac{2}{3}\mu) V(U_x + V_y + W_z)\big)_y \\ &+ \big(2\mu W W_z + \mu U(U_z + W_x) + \mu V(V_z + W_y) \;\; + \kappa T_z + (\zeta - \tfrac{2}{3}\mu) W(U_x + V_y + W_z)\big)_z \end{aligned} \tag{E.117}$$

Therein, κ denotes the thermal conductivity

$$\kappa = \mu \frac{K+5}{2} R = \rho \nu c_p = \rho k c_p, \tag{E.118}$$

where k is the thermal diffusivity and c_p is the heat capacity. Introducing the definition of the heat capacity yields a well known defect of the BGK approximation. The ratio between thermal diffusivity and kinematic viscosity is one, hence,

$$Pr = \frac{\nu}{k} = 1. \tag{E.119}$$

List of Figures

List of Tables

List of Symbols

Symbol	Dimension	Description
$a, \overline{a}, \vec{a}$	various	spatial expansion coefficients in the Taylor expansion of f_0^{eq}
a	1	concentration exponents in chemical reactions
$A, \overline{A}$	various	temporal expansion coefficients in the Taylor expansion of f_0^{eq}
A	—	pre-exponential constant in Arrhenius equation
A_j	m^2	cell face area
b	1	concentration exponents in chemical reactions
$\vec{b}$	various	velocity space expansion coefficients in the Taylor expansion of f_0^{eq}
c_D, c_L	1	drag and lift coefficients
c_v	$\mathrm{J\,kg^{-1}\,K^{-1}}$	mass specific heat capacity at constant volume
c_p	$\mathrm{J\,kg^{-1}\,K^{-1}}$	mass specific heat capacity at constant pressure
c_s	$\mathrm{m\,s^{-1}}$	speed of sound
C_S	1	Smagorinsky constant
D	$\mathrm{m^2\,s^{-1}}$	diffusivity
D	m	diameter
D_{LES}	$\mathrm{m^2\,s^{-1}}$	turbulent diffusivity
e	$\mathrm{J\,kg^{-1}}$	mass specific internal energy of a continuum
$\tilde{e}$	J	internal energy of an ensemble
$\tilde{e}_p$	J	internal energy of a particle
E	$\mathrm{J\,kg^{-1}}$	mass specific total energy
E	1	parallel efficiency
E_a	$\mathrm{J\,mol^{-1}}$	activation energy
E_{kin}	$\mathrm{J\,kg^{-1}}$	mass specific kinetic energy

$\mathcal{E}$	s^{-2}	enstrophy
f	—	scaled particle distribution function
f	s^{-1}	frequency
f^{eq}	—	scaled equilibrium distribution function
f^{neq}	—	scaled non-equilibrium part of distribution function
$f_u^{eq}, f_v^{eq}, f_w^{eq}$	—	factorization of the equilibrium distribution function
$f_0, f_1, f_2, \ldots$	—	expansion coefficients in the Chapman-Enskog expansion of f
$\tilde{f}$	—	particle distribution function
F	—	force in the Boltzmann equation
F_x, F_y	N	force on a body
$\underline{\mathcal{F}}$	various	directional flux density
$\vec{g}$	$\mathrm{m\,s}^{-2}$	gravitational acceleration
Δh	$\mathrm{J\,kg}^{-1}$	mass specific heat of combustion
H	m	height
$H(\cdot)$	1	Heaviside function
ΔH	$\mathrm{J\,mol}^{-1}$	molar heat of combustion
k	$\mathrm{m}^2\,\mathrm{s}^{-1}$	thermal diffusivity
k	s^{-1}	reaction rate
k	1	wave number
k_B	$\mathrm{J\,K}^{-1}$	Boltzmann constant
K	1	number of internal degrees of freedom
l	m	mean free path
L	m	length
m	kg	mass, ensemble mass
$\tilde{m}$	kg	particle mass
M	$\mathrm{kg\,mol}^{-1}$	molar mass
n	mol	amount of substance
$\vec{n}$	m	outward facing normal vector
N	1	ensemble size

N	1	number of GPUs
N_A	mol^{-1}	Avogadro constant
p	$\mathrm{kg\,s^{-2}\,m^{-1}}$	pressure
q	$\mathrm{J\,s^{-1}}$	heat flux
$\dot{q}'''$	$\mathrm{J\,m^{-3}\,s^{-1}}$	volumetric heat release rate
Q	J	amount of heat
$\underline{\mathcal{Q}}$	various	source term
r	1	refinement ratio
R	$\mathrm{J\,kg^{-1}\,K^{-1}}$	specific gas constant
$\underline{\underline{R}}$	1	rotation matrix
R_u	$\mathrm{J\,mol^{-1}\,K^{-1}}$	universal gas constant
s	1	stoichiometric mass ratio
S	s^{-1}	shear rate
S	1	speed up
S	K	pseudo temperature
S	m^3	solid domain
t	s	time
t^*	s	reference time
Δt	s	time step
T	K	absolute temperature
TKE	$\mathrm{J\,kg^{-1}}$	turbulent kinetic energy
$\vec{u} = (u, v, w)^T$	$\mathrm{m\,s^{-1}}$	microscopic particle velocity
$\lvert\vec{u}\rvert_p$	$\mathrm{m\,s^{-1}}$	most probable particle speed
$\delta\vec{u}$	$\mathrm{m\,s^{-1}}$	a short distance in velocity space
$\vec{U} = (U, V, W)^T$	$\mathrm{m\,s^{-1}}$	macroscopic fluid velocity
U_{max}	$\mathrm{m\,s^{-1}}$	maximal inlet velocity of the open boundary condition
U'	$\mathrm{m\,s^{-1}}$	velocity fluctuation in RANS
U''	$\mathrm{m\,s^{-1}}$	velocity fluctuation in Favre averaging
V	m^3	volume
V_i	m^3	cell volume

V_{max}	$\mathrm{m\,s^{-1}}$	maximal velocity
$\overline{V}$	$\mathrm{m\,s^{-1}}$	mean velocity
∂V	$\mathrm{m^2}$	surface of a volume V
w_0, w_1, w_2	1	interpolation coefficients for coarse to fine interpolation at the grid interface
$\underline{W} = (\rho, \rho\vec{U}, \rho E)^T$	various	tuple of conserved variables
$\underline{\tilde{W}}$	various	interpolated conserved variables at the grid interface
$\vec{x} = (x, y, z)^T$	m	coordinate vector
Δx	m	grid spacing
$\delta\vec{x}$	m	a short distance in space
X	1	mole fraction
Y	1	mass fraction
$\underline{Z} = (\rho, \vec{U}, \lambda)^T$	various	tuple of primitive variables

Greek symbols

α	1	numerical order of convergence
α	1	temperature exponent in modified Arrhenius equation
$\underline{\alpha}$	various	coefficients in the logarithm of f_0^{eq}
β	1	thermal expansion coefficient
γ	1	ratio of specific heats
ϵ	s	scale carrying part of the relaxation time
ϵ	$\mathrm{m^2\,s^{-3}}$	turbulent dissipation rate
ζ	$\mathrm{kg\,m^{-1}\,s^{-1}}$	bulk viscosity
θ	$\mathrm{m\,s^{-1}}$	internal degree of freedom related to Θ
Θ	$\mathrm{kg^{-1}}$	mass specific passive scalar
κ	$\mathrm{J\,s^{-1}\,m^{-1}\,K^{-1}}$	thermal conductivity
λ	$\mathrm{s^2\,m^{-2}}$	inverse square most probable particle speed
μ	$\mathrm{kg\,m^{-1}\,s^{-1}}$	dynamic viscosity
μ_2	$\mathrm{kg\,m^{-1}\,s^{-1}}$	second viscosity
ν	$\mathrm{m^2\,s^{-1}}$	kinematic viscosity

$\nu_{(\cdot)}$	1	stoichiometric factor
ρ	$\mathrm{kg\,m^{-3}}$	density
τ	s	relaxation time
τ	s	mixing time scale
τ_D	s	relaxation time related to the diffusivity D of the passive scalar Θ
$\hat{\tau}$	1	variation carrying part of the relaxation time
ξ	$\mathrm{m\,s^{-1}}$	coordinate of an internal degree of freedom
ξ	1	mixture fraction
ξ_{st}	1	stoichiometric mixture fraction
Ξ	$\mathrm{m^3\,s^{-3}}$	volume of the velocity space
$\underline{\psi}$	various	tuple of collision invariants
$\vec{\omega}$	$\mathrm{s^{-1}}$	vorticity
Ω	—	collision operator

Dimensionless Numbers

$Kn = l/L$	Knudsen number
$Pr = \nu/k$	Prandtl number
$Sc = \nu/D$	Schmidt number
$CFL = ((U + c_s)\Delta t)/\Delta x$	Courant-Friedrichs-Lewy number
$Ma = U/c_s$	Mach number
$Re = (U\ L)/\nu$	Reynolds number
$St = (f\ D)/U$	Strouhal number
$Ra = (Pr\ g\ H^3\ \epsilon)/\nu^2$	Rayleigh number
$Ba = (g\ H)/(R\ T_0)$	Barometric number
$\epsilon = (T_h - T_c)/T_0$	dimensionless temperature difference
$Ri = (g\ H)/U^2$	Richardson number
$Nu = (q\ L)/(\kappa_0(T_h - T_c))$	Nusselt number

Notations

$\langle\cdot\rangle$	moment of the equilibrium distribution
∇	spatial gradient

$\nabla_{\vec{u}}$	gradient in velocity space
D_t	material derivative
$(\cdot)_0$	at time initial time
$(\cdot)_0$	reference value
$(\cdot)_a$	analytic solution
$(\cdot)_{,(\cdot)}$, e.g. $(\cdot)_{,x}$	partial derivative
$(\cdot)_l$	left of the cell face
$(\cdot)_r$	right of the cell face
$(\cdot)^{(\cdot)}$, e.g. $(\cdot)^{\rho}$	component of a tuple
$(\cdot)_x$	x-component of a vector
$(\cdot)_y$	y-component of a vector
$(\cdot)_z$	z-component of a vector
$(\cdot)_i$	when applicable: of the ith cell
$(\cdot)_i$	when applicable: of the ith component
$(\cdot)_j$	when applicable: of the jth cell face
$(\cdot)'$	at an earlier time, in the context of characteristics
$(\cdot)'$	in a rotated frame of reference
$(\cdot)_0$	in the context of boundary conditions: on the boundary
$(\cdot)_1$	in the context of boundary conditions: in the first domain cell
$(\cdot)_{-1}$	in the context of boundary conditions: in the ghost cell
$(\cdot)_2$	in the context of boundary conditions: in the second domain cell
$(\cdot)_N$	in the context of boundary conditions: in the first domain cell on the opposing side of the domain
$(\cdot)_t$	modified value due to turbulence
$[\cdot]$	concentration
$\overline{(\cdot)}$	time average
$(\cdot)'$	turbulent fluctuation

$(\cdot)^{\infty}$	ambient value
$(\cdot)_F$	at the cell face center
$(\cdot)_C$	at the cell center
$(\cdot)_{CF}$	cell value extrapolated to the cell face center
$(\cdot)^{+}$	on the positive side of the interface
$(\cdot)^{-}$	on the negative side of the interface
$(\cdot)_h$	on the hot wall
$(\cdot)_c$	on the cold wall
$\widehat{(\cdot)}$	Richardson extrapolation
$\dot{(\cdot)}$	time derivative

Chemical Species

H	hydrogen
C	carbon
N	nitrogen
O	oxygen
CH_4	methane
H_2O	water
CO_2	carbon dioxide
Ox	oxidizer
F	fuel
P	products
A	air

List of Abbreviations

Abbreviation	Description
ALU	Arithmetic Logic Unit
AOS	Array of Structures
BGK	Bhatnagar-Gross-Krook
BW	Bandwidth
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CUPS	Cell Updates Per Second
DNS	Direct Numerical Simulation
DSMC	Direct Simulation Monte Carlo
DUGKS	Discrete Unified Gas Kinetic Scheme
FDS	Fire Dynamics Simulator
FTCS	Forward in Time, Central in Space
GKS	Gas Kinetic Scheme
GPU	Graphics Processing Units
GPGPU	General Purpose computing on Graphics Processing Units
LBM	Lattice Boltzmann Method
LES	Large Eddy Simulation
LU-SGS	Lower-Upper Symmetric Gauss-Seidel
MIMD	Multiple Instruction Multiple Data
MPI	Message Passing Interface
NIST	National Institute of Standards and Technology
NUMA	Non-Uniform Memory Access
NUPS	Node Updates Per Second
OpenCL	Open Compute Language
PDF	Probability Density Function
RANS	Reynolds-Averaged Navier-Stokes
SCRS	Simple Chemical Reacting System
SIMD	Single Instruction Multiple Data
SM	Streaming Machine
SOA	Structure Of Arrays
SPH	Smoothed Particle Hydrodynamics
SPMD	Single Program Multiple Data
UGKS	Unified Gas Kinetic Scheme
VTK	Visualization Toolkit
WALE	Wall-Adapting Local eddy-viscosity

WENO	Weighted Essentially Non-Oscillatory

Bibliography

[1] F. Berna, P. Goldberg, L. K. Horwitz, J. Brink, S. Holt, M. Bamford, and M. Chazan, "Microstratigraphic evidence of in situ fire in the Acheulean strata of Wonderwerk Cave, Northern Cape province, South Africa," *Proceedings of the National Academy of Sciences*, vol. 109, no. 20, pp. E1215–E1220, 2012. [Online]. Available: https://www.pnas.org/content/109/20/E1215

[2] FeuerTrutz, "Statistik der Brandtoten in Deutschland 2015," https://www.feuertrutz.de/statistik-der-brandtoten-in-deutschland-2015/150/52637/, accessed: 2019-11-18.

[3] Statistisches Bundesamt, "Ergebnisse der Todesursachenstatistik für Deutschland - Ausführliche vierstellige ICD10-Klassifikation - 2017," https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Todesursachen/Publikationen/Downloads-Todesursachen/todesursachenstatistik-5232101177015.html, 2019, accessed: 2019-11-18.

[4] M. Moore-Bick, *Grenfell Tower Inquiry: Phase 1 Report*, 2019, vol. 4. [Online]. Available: https://www.grenfelltowerinquiry.org.uk/phase-1-report

[5] K. B. McGrattan, S. Hostikka, J. Floyd, R. McDermott, and M. Vanella, *Fire Dynamics Simulator User's Guide*, 6th ed., ser. NIST Special Publication. Gaithersburg, MD: National Institute of Standards and Technology, 2006. [Online]. Available: https://github.com/firemodels/fds/releases/download/FDS6.7.3/FDS_User_Guide.pdf

[6] B. Husted, Y. Z. Li, C. Huang, J. Anderson, R. Svensson, H. Ingason, M. Runefors, and J. Wahlqvist, "Verification, validation and evaluation of FireFOAM as a tool for performance based design," https://www.brandskyddsforeningen.se/globalassets/brandforsk/rapporter-2017/brandforsk_rapport_309_131_firefoam_new.pdf, 2017, accessed: 2019-10-24.

[7] K. Xu, "A Gas-Kinetic BGK Scheme for the Navier–Stokes Equations and Its Connection with Artificial Dissipation and Godunov Method," *Journal of Computational Physics*, vol. 171, no. 1, pp. 289–335, 2001.

[8] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual review of astronomy and astrophysics*, vol. 30, no. 1, pp. 543–574, 1992.

[9] R. Benzi, S. Succi, and M. Vergassola, "The lattice boltzmann equation: theory and applications," *Physics Reports*, vol. 222, no. 3, pp. 145 – 197, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/037015739290090M

[10] S. Chen and G. D. Doolen, "LATTICE BOLTZMANN METHOD FOR FLUID FLOWS," *Annual Review of Fluid Mechanics*, vol. 30, no. 1, pp. 329–364, 1998. [Online]. Available: https://doi.org/10.1146/annurev.fluid.30.1.329

[11] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.-S. Luo, "Multiple–relaxation–time lattice Boltzmann models in three dimensions," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 360, no. 1792, pp. 437–451, Mar. 2002. [Online]. Available: https://doi.org/10.1098/rsta.2001.0955

[12] M. Geier, M. Schönherr, A. Pasquali, and M. Krafczyk, "The cumulant lattice boltzmann equation in three dimensions: Theory and validation," *Computers & Mathematics with Applications*, vol. 70, no. 4, pp. 507 – 547, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0898122115002126

[13] A. Scagliarini, L. Biferale, M. Sbragaglia, K. Sugiyama, and F. Toschi, "Lattice Boltzmann methods for thermal flows: Continuum limit and applications to compressible Rayleigh–Taylor systems," *Physics of Fluids*, vol. 22, no. 5, p. 055101, 2010.

[14] M. Su, K. Xu, and M. Ghidaoui, "Low-Speed Flow Simulation by the Gas-Kinetic Scheme," *Journal of Computational Physics*, vol. 150, no. 1, pp. 17–39, 1999.

[15] S. Lenz, M. Krafczyk, M. Geier, S. Chen, and Z. Guo, "Validation of a two-dimensional gas-kinetic scheme for compressible natural convection on structured and unstructured meshes," *International Journal of Thermal Sciences*, vol. 136, pp. 299–315, 2019.

[16] G. Ni, S. Jiang, and K. Xu, "Efficient kinetic schemes for steady and unsteady flow simulations on unstructured meshes," *Journal of Computational Physics*, vol. 227, no. 6, pp. 3015–3031, 2008.

[17] S. Xiong, C. Zhong, C. Zhuo, K. Li, X. Chen, and J. Cao, "Numerical simulation of compressible turbulent flow via improved gas-kinetic BGK scheme," *International Journal for Numerical Methods in Fluids*, vol. 67, no. 12, pp. 1833–1847, 2011.

[18] W. Li, M. Kaneda, and K. Suga, "An implicit gas kinetic BGK scheme for high temperature equilibrium gas flows on unstructured meshes," *Computers & Fluids*, vol. 93, pp. 100–106, 2014.

[19] D. Pan, C. Zhong, J. Li, and C. Zhuo, "A gas-kinetic scheme for the simulation of turbulent flows on unstructured meshes," *International Journal for Numerical Methods in Fluids*, vol. 82, no. 11, pp. 748–769, 2016.

[20] iRMB, "VirtualFluids," https://www.tu-braunschweig.de/irmb/forschung/virtualfluids, Institute for computational modeling in civil engineering of the Technische Universität Braunschweig, Accessed: 2019-11-15.

[21] D. Hänel, *Molekulare Gasdynamik*. Berlin/Heidelberg: Springer-Verlag, 2004.

[22] Y.-C. Cheng, *Macroscopic and Statistical Thermodynamics*. WORLD SCIENTIFIC, 2006.

[23] B. Güttler, O. Rienitz, and A. Pramann, "The Avogadro Constant for the Definition and Realization of the Mole," *Annalen der Physik*, vol. 531, no. 5, p. 1800292, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.201800292

[24] National Institute of Standards and Technology, "NIST-JANAF Thermochemical Tables," https://janaf.nist.gov/, accessed: 2019-11-21.

[25] G. Wedler, *Lehrbuch der physikalischen Chemie*, 4th ed. Weinheim: Wiley-VCH, 1997.

[26] K. Xu, *Direct modeling for computational fluid dynamics: Construction and application of unified gas-kinetic schemes*, ser. Advances in computational fluid dynamics. Singapore and Hackensack, N.J: World Scientific Pub. Co, 2015, vol. vol. 4.

[27] A. A. Mohamad, *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*. London: Springer-Verlag London Limited, 2011.

[28] D. Wolf-Gladrow, *Lattice gas cellular automata and lattice Boltzmann models: An introduction*, ser. Lecture notes in mathematics. Berlin: Springer, 2002, vol. 1725.

[29] J. C. Maxwell, "IV. On the dynamical theory of gases," *Philosophical Transactions of the Royal Society of London*, vol. 157, pp. 49–88, 1867.

[30] L. Boltzmann, "Weitere Studien über das Wärmegleichgewicht unter Gasmolekülen," *Sitzungsberichte Akademie der Wissenschaften*, vol. 66, pp. 275–370, 1872.

[31] H. A. Jakobsen, *Chemical reactor modeling: Multiphase reactive flows*. Cham: Springer, 2014.

[32] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems," *Physical Review*, vol. 94, no. 3, pp. 511–525, 1954.

[33] M. N. Kogan, *Rarefied Gas Dynamics*. Boston, MA and s.l.: Springer US, 1969.

[34] K. Xu, "Gas-Kinetic Schemes for Unsteady Compressible Flow Simulations," *von Karman Institute for Fluid Dynamics Lecture Series*, no. 3, 1998. [Online]. Available: http://www.math.ust.hk/~makxu/PAPER/vki_xu.pdf

[35] X. He and L.-S. Luo, "A priori derivation of the lattice Boltzmann equation," *Physical Review E*, vol. 55, no. 6, pp. R6333–R6336, 1997.

[36] C. T. Tian, K. Xu, K. L. Chan, and L. C. Deng, "A three-dimensional multidimensional gas-kinetic scheme for the Navier–Stokes equations under gravitational fields," *Journal of Computational Physics*, vol. 226, no. 2, pp. 2003–2027, 2007.

[37] R. W. Johnson, *The handbook of fluid dynamics*. Heidelberg and Boca Raton: Springer and CRC Press, 1998.

[38] P. Lallemand and F. Dubois, "Comparison of Simulations of Convective Flows," *Communications in Computational Physics*, vol. 17, no. 05, pp. 1169–1184, 2015.

[39] Y. Feng, P. Sagaut, and W. Tao, "A three dimensional lattice model for thermal compressible flow on standard lattices," *Journal of Computational Physics*, vol. 303, pp. 514–529, 2015.

[40] G. A. Bird, "Monte Carlo Simulation of Gas Flows," *Annual Review of Fluid Mechanics*, vol. 10, no. 1, pp. 11–31, 1978.

[41] J.-C. Huang, K. Xu, and P. Yu, "A Unified Gas-Kinetic Scheme for Continuum and Rarefied Flows II: Multi-Dimensional Cases," *Communications in Computational Physics*, vol. 12, no. 3, pp. 662–690, 2012.

[42] R. H. Sanders and K. H. Prendergast, "The Possible Relation of the 3-KILOPARSEC Arm to Explosions in the Galactic Nucleus," *The Astrophysical Journal*, vol. 188, p. 489, 1974.

[43] K. Xu, "Numerical hydrodynamics from gas-kinetic theory," Ph.D. thesis, Columbia University, 1993. [Online]. Available: http://www.math.ust.hk/~makxu/papers.html

[44] K. H. Prendergast and K. Xu, "Numerical Hydrodynamics from Gas-Kinetic Theory," *Journal of Computational Physics*, vol. 109, no. 1, pp. 53–66, 1993.

[45] K. Xu and K. H. Prendergast, "Numerical Navier-Stokes Solutions from Gas Kinetic Theory," *Journal of Computational Physics*, vol. 114, no. 1, pp. 9–17, 1994.

[46] B. van Leer, "Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection," *Journal of Computational Physics*, vol. 23, no. 3, pp. 276–299, 1977.

[47] K. Xu, M. Mao, and L. Tang, "A multidimensional gas-kinetic BGK scheme for hypersonic viscous flow," *Journal of Computational Physics*, vol. 203, no. 2, pp. 405–421, 2005.

[48] S. Yoon and A. Jameson, "Lower-upper Symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations," *AIAA Journal*, vol. 26, no. 9, pp. 1025–1026, 1988.

[49] Q. Li and S. Fu, "On the multidimensional gas-kinetic BGK scheme," *Journal of Computational Physics*, vol. 220, no. 1, pp. 532–548, 2006.

[50] T. Ohwada, "On the Construction of Kinetic Schemes," *Journal of Computational Physics*, vol. 177, no. 1, pp. 156–175, 2002.

[51] G. May, B. Srinivasan, and A. Jameson, "An improved gas-kinetic BGK finite-volume method for three-dimensional transonic flow," *Journal of Computational Physics*, vol. 220, no. 2, pp. 856–878, 2007.

[52] K. Xu, "A Slope-Update Scheme for Compressible Flow Simulation," *Journal of Computational Physics*, vol. 178, no. 1, pp. 252–259, 2002.

[53] W. Liao, Y. Peng, and L.-S. Luo, "Gas-kinetic schemes for direct numerical simulations of compressible homogeneous turbulence," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 80, no. 4 Pt 2, p. 046702, 2009.

[54] G. Kumar, S. S. Girimaji, and J. Kerimo, "WENO-enhanced gas-kinetic scheme for direct simulations of compressible transition and turbulence," *Journal of Computational Physics*, vol. 234, pp. 499–523, 2013.

[55] Q. Li, S. Tan, S. Fu, and K. Xu, "Numerical simulation of compressible turbulence with gas-kinetic BGK scheme," *Proceedings of the 13th Asian Congress of Fluid Mechanics*, pp. 1109–1112, 2010.

[56] M. Righi, "A Modified Gas-Kinetic Scheme for Turbulent Flow," *Communications in Computational Physics*, vol. 16, no. 01, pp. 239–263, 2014.

[57] J. Li, C. Zhong, D. Pan, and C. Zhuo, "A gas-kinetic scheme coupled with SST model for turbulent flows," *Computers & Mathematics with Applications*, vol. 78, no. 4, pp. 1227–1242, 2019.

[58] M. Righi, "A Gas-Kinetic Scheme for Turbulent Flow," *Flow, Turbulence and Combustion*, vol. 97, no. 1, pp. 121–139, 2016.

[59] Q. Li and S. Fu, "A High-Order Accurate Gas-Kinetic BGK Scheme," in *Computational Fluid Dynamics*, Choi H., Choi H.G., and Yoo J.Y., Eds. Berlin, Heidelberg: Springer, 2008.

[60] Q. Li, K. Xu, and S. Fu, "A high-order gas-kinetic Navier–Stokes flow solver," *Journal of Computational Physics*, vol. 229, no. 19, pp. 6715–6731, 2010.

[61] J. Luo and K. Xu, "A high-order multidimensional gas-kinetic scheme for hydrodynamic equations," *Science China Technological Sciences*, vol. 56, no. 10, pp. 2370–2384, 2013.

[62] N. Liu and H. Tang, "A High-Order Accurate Gas-Kinetic Scheme for One- and Two-Dimensional Flow Simulation," *Communications in Computational Physics*, vol. 15, no. 4, pp. 911–943, 2014.

[63] L. Pan and K. Xu, "A Compact Third-Order Gas-Kinetic Scheme for Compressible Euler and Navier-Stokes Equations," *Communications in Computational Physics*, vol. 18, no. 4, pp. 985–1011, 2015.

[64] L. Pan and K. Xu, "A third-order compact gas-kinetic scheme on unstructured meshes for compressible Navier–Stokes solutions," *Journal of Computational Physics*, vol. 318, pp. 327–348, 2016.

[65] L. Pan, K. Xu, Q. Li, and J. Li, "An efficient and accurate two-stage fourth-order gas-kinetic scheme for the Euler and Navier–Stokes equations," *Journal of Computational Physics*, vol. 326, pp. 197–221, 2016.

[66] L. Pan and K. Xu, "Two-stage fourth-order gas-kinetic scheme for three-dimensional Euler and Navier-Stokes solutions," *International Journal of Computational Fluid Dynamics*, vol. 32, no. 10, pp. 395–411, 2018.

[67] X. Ji, F. Zhao, W. Shyy, and K. Xu, "A family of high-order gas-kinetic schemes and its comparison with Riemann solver based high-order methods," *Journal of Computational Physics*, vol. 356, pp. 150–173, 2018.

[68] G. Cao, H. Su, J. Xu, and K. Xu, "Implicit high-order gas kinetic scheme for turbulence simulation," *Aerospace Science and Technology*, vol. 92, pp. 958–971, 2019.

[69] F. Zhao, X. Ji, W. Shyy, and K. Xu, "Compact higher-order gas-kinetic schemes with spectral-like resolution for compressible flow simulations," *Advances in Aerodynamics*, vol. 1, no. 1, p. 63, 2019.

[70] J. Luo, K. Xu, and N. Liu, "A Well-Balanced Symplecticity-Preserving Gas-Kinetic Scheme for Hydrodynamic Equations under Gravitational Field," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2356–2381, 2011.

[71] S. Chen, Z. Guo, and K. Xu, "A well-balanced gas kinetic scheme for Navier-Stokes equations with gravitational potential," 2019. [Online]. Available: https://arxiv.org/abs/1904.00178

[72] L.-J. Xuan and K. Xu, "A new gas-kinetic scheme based on analytical solutions of the BGK equation," *Journal of Computational Physics*, vol. 234, pp. 524–539, 2013.

[73] N. Parashar, B. Srinivasan, S. S. Sinha, and M. Agarwal, "GPU-accelerated direct numerical simulations of decaying compressible turbulence employing a GKM-based solver," *International Journal for Numerical Methods in Fluids*, vol. 83, no. 10, pp. 737–754, 2017.

[74] K. Xu and E. Josyula, "Gas-kinetic scheme for rarefied flow simulation," *Mathematics and Computers in Simulation*, vol. 72, no. 2-6, pp. 253–256, 2006.

[75] K. Xu and J.-C. Huang, "A unified gas-kinetic scheme for continuum and rarefied flows," *Journal of Computational Physics*, vol. 229, no. 20, pp. 7747–7764, 2010.

[76] K. Xu and J.-C. Huang, "An improved unified gas-kinetic scheme and the study of shock structures," *IMA Journal of Applied Mathematics*, vol. 76, no. 5, pp. 698–711, 2011.

[77] J.-C. Huang, K. Xu, and P. Yu, "A Unified Gas-Kinetic Scheme for Continuum and Rarefied Flows III: Microflow Simulations," *Communications in Computational Physics*, vol. 14, no. 5, pp. 1147–1173, 2013.

[78] C. Liu and K. Xu, "A Unified Gas Kinetic Scheme for Continuum and Rarefied Flows V: Multiscale and Multi-Component Plasma Transport," *Communications in Computational Physics*, vol. 22, no. 5, pp. 1175–1223, 2017.

[79] Y. Zhu, C. Zhong, and K. Xu, "Unified gas-kinetic scheme with multigrid convergence for rarefied flow study," *Physics of Fluids*, vol. 29, no. 9, p. 096102, 2017.

[80] Z. Guo, K. Xu, and R. Wang, "Discrete unified gas kinetic scheme for all Knudsen number flows: Low-speed isothermal case," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 88, no. 3, p. 033305, 2013.

[81] Z. Guo, R. Wang, and K. Xu, "Discrete unified gas kinetic scheme for all Knudsen number flows. II. Thermal compressible case," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 91, no. 3, p. 033313, 2015.

[82] L. Zhu, Z. Guo, and K. Xu, "Discrete unified gas kinetic scheme on unstructured meshes," *Computers & Fluids*, vol. 127, pp. 211–225, 2016.

[83] Y. Bo, P. Wang, Z. Guo, and L.-P. Wang, "DUGKS simulations of three-dimensional Taylor–Green vortex flow and turbulent channel flow," *Computers & Fluids*, vol. 155, pp. 9–21, 2017.

[84] S. Tao, H. Zhang, Z. Guo, and L.-P. Wang, "A combined immersed boundary and discrete unified gas kinetic scheme for particle–fluid flows," *Journal of Computational Physics*, vol. 375, pp. 498–518, 2018.

[85] C. Zhang, K. Yang, and Z. Guo, "A discrete unified gas-kinetic scheme for immiscible two-phase flows," *International Journal of Heat and Mass Transfer*, vol. 126, pp. 1326–1336, 2018.

[86] K. Xu and S. H. Lui, "Rayleigh-Bénard simulation using the gas-kinetic Bhatnagar-Gross-Krook scheme in the incompressible limit," *Physical Review E*, vol. 60, no. 1, pp. 464–470, 1999.

[87] C. Jin, K. Xu, and S. Chen, "A Three Dimensional Gas-Kinetic Scheme with Moving Mesh for Low-Speed Viscous Flow Computations," *Advances in Applied Mathematics and Mechanics*, vol. 2, no. 6, pp. 746–762, 2010.

[88] S. Chen, C. Jin, C. Li, and Q. Cai, "Gas-kinetic scheme with discontinuous derivative for low speed flow computation," *Journal of Computational Physics*, vol. 230, no. 5, pp. 2045–2059, 2011.

[89] P. Wang and Z. Guo, "A semi-implicit gas-kinetic scheme for smooth flows," *Computer Physics Communications*, vol. 205, pp. 22–31, 2016.

[90] D. Zhou, Z. Lu, and T. Guo, "A Gas-Kinetic BGK Scheme for Natural Convection in a Rotating Annulus," *Applied Sciences*, vol. 8, no. 5, p. 733, 2018.

[91] C. Li and L.-P. Wang, "An immersed boundary-discrete unified gas kinetic scheme for simulating natural convection involving curved surfaces," *International Journal of Heat and Mass Transfer*, vol. 126, pp. 1059–1070, 2018.

[92] P. Wang, S. Tao, and Z. Guo, "A coupled discrete unified gas-kinetic scheme for Boussinesq flows," *Computers & Fluids*, vol. 120, pp. 70–81, 2015.

[93] P. Wang, Y. Zhang, and Z. Guo, "Numerical study of three-dimensional natural convection in a cubical cavity at high Rayleigh numbers," *International Journal of Heat and Mass Transfer*, vol. 113, pp. 217–228, 2017.

[94] K. Xu and X. He, "Lattice Boltzmann method and gas-kinetic BGK scheme in the low-Mach number viscous flow simulations," *Journal of Computational Physics*, vol. 190, no. 1, pp. 100–117, 2003.

[95] Z. Guo, H. Liu, L.-S. Luo, and K. Xu, "A comparative study of the LBE and GKS methods for 2D near incompressible laminar flows," *Journal of Computational Physics*, vol. 227, no. 10, pp. 4955–4976, 2008.

[96] P. Wang, L.-P. Wang, and Z. Guo, "Comparison of the lattice Boltzmann equation and discrete unified gas-kinetic scheme methods for direct numerical simulation of decaying turbulent flows," *Physical review. E*, vol. 94, no. 4-1, p. 043304, 2016.

[97] S. Lenz, M. Geier, and M. Krafczyk, "An explicit gas kinetic scheme algorithm on non-uniform Cartesian meshes for GPGPU architectures," *Computers & Fluids*, vol. 186, pp. 58–73, 2019.

[98] K. Xu, "BGK-Based Scheme for Multicomponent Flow Calculations," *Journal of Computational Physics*, vol. 134, no. 1, pp. 122–133, 1997.

[99] Y. S. Lian and K. Xu, "A Gas-Kinetic Scheme for Multimaterial Flows and Its Application in Chemical Reactions," *Journal of Computational Physics*, vol. 163, no. 2, pp. 349–375, 2000.

[100] Q. Li, S. Fu, and K. Xu, "A compressible Navier–Stokes flow solver with scalar transport," *Journal of Computational Physics*, vol. 204, no. 2, pp. 692–714, 2005.

[101] J. Blazek, *Computational fluid dynamics: Principles and applications*, third edition ed. Amsterdam and San Diego: Butterworth Heinemann, 2015.

[102] J. H. Ferziger and M. Perić, *Numerische Strömungsmechanik*. Berlin: Springer, 2008.

[103] P. Sagaut, *Large Eddy Simulation for Incompressible Flows*. Berlin/Heidelberg: Springer-Verlag, 2006.

[104] U. Frisch, *Turbulence*. Cambridge University Press, 2018.

[105] P. A. Davidson, *Turbulence: An introduction for scientists and engineers*. Oxford, UK and New York: Oxford University Press, 2010.

[106] J. Smagorinsky, "GENERAL CIRCULATION EXPERIMENTS WITH THE PRIMITIVE EQUATIONS," *Monthly Weather Review*, vol. 91, no. 3, pp. 99–164, 1963.

[107] K. B. McGrattan, S. Hostikka, J. Floyd, R. McDermott, and M. Vanella, *Fire Dynamics Simulator Technical Reference Guide: Volume 3: Validation*, 6th ed., ser. NIST Special Publication. Gaithersburg, MD: National Institute of Standards and Technology, 2006. [Online]. Available: https://github.com/firemodels/fds/releases/download/FDS6.7.3/FDS_Technical_Reference_Guide.pdf

[108] K. Lilly, "The representation of small-scale turbulence in numerical simulation experiments," 1966.

[109] O. Filippova and D. Hänel, "Grid Refinement for Lattice-BGK Models," *Journal of Computational Physics*, vol. 147, no. 1, pp. 219–228, 1998.

[110] M. Geier, A. Greiner, and J. G. Korvink, "Bubble functions for the lattice Boltzmann method and their application to grid refinement," *The European Physical Journal Special Topics*, vol. 171, no. 1, pp. 173–179, 2009.

[111] S. Lenz, M. Schönherr, M. Geier, M. Krafczyk, A. Pasquali, A. Christen, and M. Giometto, "Towards real-time simulation of turbulent air flow over a resolved urban canopy using the cumulant lattice Boltzmann method on a GPGPU," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 189, pp. 151–162, 2019.

[112] M. A. Olshanskii, K. M. Terekhov, and Y. V. Vassilevski, "An octree-based solver for the incompressible Navier–Stokes equations with enhanced stability and low dissipation," *Computers & Fluids*, vol. 84, pp. 231–246, 2013.

[113] A. Pletzer, B. Jamroz, R. Crockett, and S. Sides, "Compact cell-centered discretization stencils at fine–coarse block structured grid interfaces," *Journal of Computational Physics*, vol. 260, pp. 25–36, 2014.

[114] A. Guittet, M. Theillard, and F. Gibou, "A stable projection method for the incompressible Navier–Stokes equations on arbitrary geometries and adaptive Quad/Octrees," *Journal of Computational Physics*, vol. 292, pp. 215–238, 2015.

[115] C. Batty, "A cell-centred finite volume method for the Poisson problem on non-graded quadtrees with second order accurate gradients," *Journal of Computational Physics*, vol. 331, pp. 49–72, 2017.

[116] R. Yuan, C. Zhong, and H. Zhang, "An immersed-boundary method based on the gas kinetic BGK scheme for incompressible viscous flow," *Journal of Computational Physics*, vol. 296, pp. 184–208, 2015.

[117] F. Lyu, T. Xiao, and X. Yu, "A fast and automatic full-potential finite volume solver on Cartesian grids for unconventional configurations," *Chinese Journal of Aeronautics*, vol. 30, no. 3, pp. 951–963, 2017.

[118] R. Courant, K. Friedrichs, and H. Lewy, "Über die partiellen differenzengleichungen der mathematischen physik," *Mathematische Annalen*, vol. 100, no. 1, pp. 32–74, Dec 1928. [Online]. Available: https://doi.org/10.1007/BF01448839

[119] U. Ghia, K. Ghia, and C. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, no. 3, pp. 387–411, 1982.

[120] J. G. Quintiere, *Principles of Fire Behavior, Second Edition.* CRC Press, 2016.

[121] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: The finite volume method*, 2nd ed. Harlow: Pearson/Prentice Hall, 2007.

[122] S. K. Upadhyay, *Chemical Kinetics and Reaction Dynamics.* Dordrecht: Springer Netherlands, 2006.

[123] D. B. Spalding, "Mixing and chemical reaction in steady confined turbulent flames," *Symposium (International) on Combustion*, vol. 13, no. 1, pp. 649–657, 1971.

[124] P. E. DesJardin and S. H. Frankel, "Large eddy simulation of a nonpremixed reacting jet: Application and assessment of subgrid-scale combustion models," *Physics of Fluids*, vol. 10, no. 9, pp. 2298–2314, 1998.

[125] M. Sheikhi, T. G. Drozda, P. Givi, F. A. Jaberi, and S. B. Pope, "Large eddy simulation of a turbulent nonpremixed piloted methane jet flame (Sandia Flame D)," *Proceedings of the Combustion Institute*, vol. 30, no. 1, pp. 549–556, 2005.

[126] W. Malalasekera, S. S. Ibrahim, A. R. Masri, S. R. Gubba, and S. Sadasivuni, "Experience With the Large Eddy Simulation (LES) Technique for the Modeling of Premixed and Non-Premixed Combustion," *Heat Transfer Engineering*, vol. 34, no. 14, pp. 1156–1170, 2013.

[127] U. Wickström, "Temperature Calculation in Fire Safety Engineering," 2016.

[128] R. C. Till and J. W. Coon, *Fire Protection.* Cham: Springer International Publishing, 2019.

[129] K. B. McGrattan, S. Hostikka, J. Floyd, R. McDermott, and M. Vanella, *Fire Dynamics Simulator Technical Reference Guide: Volume 2: Verification*, 6th ed., ser. NIST Special Publication. Gaithersburg, MD: National Institute of Standards and Technology, 2006. [Online]. Available: https://github.com/firemodels/fds/releases/download/FDS6.7.3/FDS_Verification_Guide.pdf

[130] K. B. McGrattan, S. Hostikka, J. Floyd, R. McDermott, and M. Vanella, *Fire Dynamics Simulator Technical Reference Guide: Volume 3: Validation*, 6th ed., ser. NIST Special Publication. Gaithersburg, MD: National Institute of Standards and Technology, 2006. [Online]. Available: https://github.com/firemodels/fds/releases/download/FDS6.7.3/FDS_Validation_Guide.pdf

[131] B. F. Magnussen and B. H. Hjertager, "On mathematical modeling of turbulent combustion with special emphasis on soot formation and combustion," *Symposium (International) on Combustion*, vol. 16, no. 1, pp. 719–729, 1977.

[132] K. McGrattan, S. Hostikka, J. Floyd, H. Baum, and R. Rehm, *Fire Dynamics Simulator (Version 5): Technical Reference Guide*, ser. NIST Special Publication. United States: National Institute of Standards and Technology NIST, 2007. [Online]. Available: https://www.vtt.fi/inf/julkaisut/muut/2007/NIST_SP_1018_5.pdf

[133] Y. Xin, J. Gore, K. McGrattan, R. Rehm, and H. Baum, "Fire dynamics simulation of a turbulent buoyant flame using a mixture-fraction-based combustion model," *Combustion and Flame*, vol. 141, no. 4, pp. 329–335, 2005.

[134] S. Geller, M. Krafczyk, J. Tölke, S. Turek, and J. Hron, "Benchmark computations based on lattice-Boltzmann, finite element and finite volume methods for laminar flows," *Computers & Fluids*, vol. 35, no. 8-9, pp. 888–897, 2006.

[135] J. Tölke, S. Freudiger, and M. Krafczyk, "An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations," *Computers & Fluids*, vol. 35, no. 8-9, pp. 820–830, 2006.

[136] J. Tölke and M. Krafczyk, "TeraFLOP computing on a desktop PC with GPUs for 3D CFD," *International Journal of Computational Fluid Dynamics*, vol. 22, no. 7, pp. 443–456, 2008.

[137] M. Schönherr, K. Kucher, M. Geier, M. Stiebler, S. Freudiger, and M. Krafczyk, "Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs," *Computers & Mathematics with Applications*, vol. 61, no. 12, pp. 3730–3743, 2011.

[138] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, p. 40, 2008.

[139] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–72, 2010.

[140] D. B. Kirk and W. W. Hwu, *Programming massively parallel processors: A hands-on approach*, 2nd ed. Amsterdam: Elsevier Morgan Kaufmann, 2013.

[141] NVIDIA, "NVIDIA Tesla P100: Whitepaper," 2016. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[142] MPI Forum, "MPI: Message Passing Interface," https://www.mpi-forum.org/, accessed: 2019-11-21.

[143] W. Schroeder, K. Martin, and B. Lorensen, *The visualization toolkit: An object-oriented approach to 3D graphics*, 4th ed. Clifton Park, NY: Kitware Inc, 2006. [Online]. Available: https://www.kitware.com/products/books/VTKTextbook.pdf

[144] U. Ayachit, *The ParaView guide: Updated for ParaView version 4.3*, full color version ed. Los Alamos: Kitware, 2015. [Online]. Available: https://www.paraview.org/paraview-guide/

[145] T. Deakin, J. Price, M. Martineau, and S. McIntosh-Smith, "GPU-STREAM v2.0: Benchmarking the Achievable Memory Bandwidth of Many-Core Processors Across Diverse Parallel Programming Models," in *High Performance Computing*, ser. Lecture Notes in Computer Science, M. Taufer, B. Mohr, and J. M. Kunkel, Eds., vol. 9945. Cham: Springer International Publishing, 2016, pp. 489–507.

[146] A. Pasquali, "Enabling the cumulant lattice Boltzmann method for complex CFD engineering problems," Dissertation, TU Braunschweig, 2017. [Online]. Available: http://publikationsserver.tu-braunschweig.de/get/64219

[147] OpenFOAM Foundation, "OpenFOAM," https://www.openfoam.com/, accessed: 2019-11-21.

[148] S. Peters, "Entwicklung eines skalierbaren massiv parallelen Gittergenerators unter Einsatz von GPGPUs," Master thesis, TU Braunschweig, 2017.

[149] M. Schönherr, "Towards reliable LES-CFD computations based on advanced LBM models utilizing (Multi-) GPGPU hardware," Dissertation, TU Braunschweig, 2015. [Online]. Available: http://www.digibib.tu-bs.de/?docid=00062945

[150] D. Lagrava, O. Malaspinas, J. Latt, and B. Chopard, "Advances in multi-domain lattice Boltzmann grid refinement," *Journal of Computational Physics*, vol. 231, no. 14, pp. 4808–4822, 2012.

[151] M. Geier and M. Schönherr, "Esoteric Twist: An Efficient in-Place Streaming Algorithmus for the Lattice Boltzmann Method on Massively Parallel Hardware," *Computation*, vol. 5, no. 2, 2017. [Online]. Available: https://www.mdpi.com/2079-3197/5/2/19

[152] S. Bindick, "Ein integrierter Ansatz zur interaktiven dreidimensionalen Simulation gekoppelter thermischer Prozesse," Dissertation, TU Braunschweig, 2010. [Online]. Available: http://www.digibib.tu-bs.de/?docid=00037192

[153] S. Freudiger, "Entwicklung eines parallelen, adaptiven, komponentenbasierten Strömungskerns für hierarchische Gitter auf Basis des Lattice-Boltzmann-Verfahrens," Dissertation, TU Braunschweig, 2009. [Online]. Available: http://www.digibib.tu-bs.de/?docid=00029863

[154] G. Magnus, "Ueber die Abweichung der Geschosse, und: Ueber eine auffallende Erscheinung bei rotirenden Körpern," *Annalen der Physik*, vol. 164, no. 1, pp. 1–29, 1853. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.18531640102

[155] A. I. Heft, T. Indinger, and N. A. Adams, "Introduction of a new realistic generic car model for aerodynamic investigations," in *SAE 2012 World Congress & Exhibition*. SAE International, apr 2012. [Online]. Available: https://doi.org/10.4271/2012-01-0168

[156] G. Karypis and V. Kumar, "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0," http://www.cs.umn.edu/~metis, University of Minnesota, Minneapolis, MN, 2009.

[157] P. Tucker and Z. Pan, "A Cartesian cut cell method for incompressible viscous flow," *Applied Mathematical Modelling*, vol. 24, no. 8-9, pp. 591–606, 2000.

[158] D. M. Ingram, D. M. Causon, and C. G. Mingham, "Developments in Cartesian cut cell methods," *Mathematics and Computers in Simulation*, vol. 61, no. 3-6, pp. 561–572, 2003.

[159] D. Hartmann, M. Meinke, and W. Schröder, "An adaptive multilevel multigrid formulation for Cartesian hierarchical grid methods," *Computers & Fluids*, vol. 37, no. 9, pp. 1103–1125, 2008.

[160] L. Schneiders, D. Hartmann, M. Meinke, and W. Schröder, "An accurate moving boundary formulation in cut-cell methods," *Journal of Computational Physics*, vol. 235, pp. 786–809, 2013.

[161] T. Ye, R. Mittal, H. S. Udaykumar, and W. Shyy, "An Accurate Cartesian Grid Method for Viscous Incompressible Flows with Complex Immersed Boundaries," *Journal of Computational Physics*, vol. 156, no. 2, pp. 209–240, 1999.

[162] M. P. Kirkpatrick, S. W. Armfield, and J. H. Kent, "A representation of curved boundaries for the solution of the Navier–Stokes equations on a staggered three-dimensional Cartesian grid," *Journal of Computational Physics*, vol. 184, no. 1, pp. 1–36, 2003.

[163] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, "Benchmark Computations of Laminar Flow Around a Cylinder," in *Flow Simulation with High-Performance Computers II*, E. H. Hirschel, K. Fujii, B. van Leer, M. A. Leschziner, M. Pandolfi, A. Rizzi, and B. Roux, Eds. Wiesbaden: Vieweg+Teubner Verlag, 1996, vol. 48, pp. 547–566.

[164] G. de Vahl Davis, "Natural convection of air in a square cavity: A bench mark numerical solution," *International Journal for Numerical Methods in Fluids*, vol. 3, no. 3, pp. 249–264, 1983.

[165] P. Le Quéré, C. Weisman, H. Paillère, J. Vierendeels, E. Dick, R. Becker, M. Braack, and J. Locke, "Modelling of Natural Convection Flows with Large Temperature Differences: A Benchmark Problem for Low Mach Number Solvers. Part 1. Reference Solutions," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 39, no. 3, pp. 609–616, 2005.

[166] H. Paillère, P. Le Quéré, C. Weisman, J. Vierendeels, E. Dick, M. Braack, F. Dabbene, A. Beccantini, E. Studer, T. Kloczko, C. Corre, V. Heuveline, M. Darbandi, and S. F. Hosseinizadeh, "Modelling of Natural Convection Flows with Large Temperature Differences: A Benchmark Problem for Low Mach Number Solvers. Part 2. Contributions to the June 2004 conference," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 39, no. 3, pp. 617–621, 2005.

[167] R. Becker and M. Braack, "Solution of a stationary benchmark problem for natural convection with large temperature difference," *International Journal of Thermal Sciences*, vol. 41, no. 5, pp. 428–439, 2002.

[168] J. Vierendeels, B. Merci, and E. Dick, "Numerical study of natural convective heat transfer with large temperature differences," *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 11, no. 4, pp. 329–341, 2001.

[169] J. Vierendeels, B. Merci, and E. Dick, "Benchmark solutions for the natural convective heat transfer problem in a square cavity with large horizontal temperature differences," *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 13, no. 8, pp. 1057–1078, 2003.

[170] W. Sutherland, "LII. The viscosity of gases and molecular force," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 36, no. 223, pp. 507–531, 1893.

[171] E. P. van der Poel, R. J. A. M. Stevens, and D. Lohse, "Comparison between two and three dimensional Rayleigh-Bénard convection," *Journal of Fluid Mechanics*, vol. 736, pp. 177–194, 2013.

[172] Y. Bao, J. Luo, and M. Ye, "Parallel Direct Method of DNS for Two-Dimensional Turbulent Rayleigh-Bénard Convection," *Journal of Mechanics*, vol. 66, pp. 1–8, 2017.

[173] P. Tabeling, "Two-dimensional turbulence: A physicist approach," *Physics Reports*, vol. 362, no. 1, pp. 1–62, 2002.

[174] M. Geier, S. Lenz, M. Schönherr, and M. Krafczyk, "Implicit and explicit large eddy simulations of a decaying Taylor Green vortex with cumulant lattice Boltzmann and gas kinetic scheme," *submitted to Physical Review E*.

[175] G. I. Taylor and B. A. Green, "Mechanism of the production of small eddies from large ones," *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences*, vol. 158, no. 895, pp. 499–521, 1937.

[176] R. H. Kraichnan, "Inertial Ranges in Two-Dimensional Turbulence," *Physics of Fluids*, vol. 10, no. 7, p. 1417, 1967.

[177] Z. J. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. T. Huynh, N. Kroll, G. May, P.-O.

Persson, B. van Leer, and M. Visbal, "High-order CFD methods: Current status and perspective," *International Journal for Numerical Methods in Fluids*, vol. 72, no. 8, pp. 811–845, 2013.

[178] C. T. Jacobs, S. P. Jammy, and N. D. Sandham, "OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures," *Journal of Computational Science*, vol. 18, pp. 12–23, 2017.

[179] D. Foti and K. Duraisamy, "An investigation of an implicit large-eddy simulation framework for the vorticity transport equations," in *2018 Fluid Dynamics Conference*, 2018, p. 3407.

[180] M. Geier, A. Pasquali, and M. Schönherr, "Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part I: Derivation and validation," *Journal of Computational Physics*, vol. 348, pp. 862 – 888, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999117304230

[181] M. Geier, A. Pasquali, and M. Schönherr, "Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part II: Application to flow around a sphere at drag crisis," *Journal of Computational Physics*, vol. 348, pp. 889 – 898, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999117305065

[182] M. Geier and A. Pasquali, "Fourth order Galilean invariance for the lattice Boltzmann method," *Computers & Fluids*, vol. 166, pp. 139 – 151, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045793018300239

[183] Felix Dietzsch, "Matlab code for the spectral analysis of homogenous isotropic turbulence," https://github.com/fdietzsc/hita, 2012, accessed: 2019-11-05.

[184] Y. S. Tian and T. G. Karayiannis, "Low turbulence natural convection in an air filled square cavity: Part I: the thermal and fluid flow fields," *International Journal of Heat and Mass Transfer*, vol. 43, no. 6, pp. 849–866, 2000.

[185] Y. S. Tian and T. G. Karayiannis, "Low turbulence natural convection in an air filled square cavity: Part II: the turbulence quantities," *International Journal of Heat and Mass Transfer*, vol. 43, no. 6, pp. 867–884, 2000.

[186] M. Salinas-Vázquez, W. Vicente, E. Martínez, and E. Barrios, "Large eddy simulation of a confined square cavity with natural convection based on compressible flow equations," *International Journal of Heat and Fluid Flow*, vol. 32, no. 5, pp. 876–888, 2011.

[187] S. R. Tieszen, T. J. O'Hern, R. W. Schefer, E. J. Weckman, and T. K. Blanchat, "Experimental study of the flow field in and around a one meter diameter methane fire," *Combustion and Flame*, vol. 129, no. 4, pp. 378–391, 2002.

[188] S. R. Tieszen, T. J. O'Hern, E. J. Weckman, and R. W. Schefer, "Experimental study of the effect of fuel mass flux on a 1-m-diameter methane fire and comparison with a hydrogen fire," *Combustion and Flame*, vol. 139, no. 1-2, pp. 126–141, 2004.

[189] S. Lenz, "Development of efficient Data Structures and Validation Studies for two-dimensional, compressible low Ma-flows for the Gas Kinetic Scheme," Master thesis, TU Braunschweig, 2017.

[190] H. Schlichting and K. Gersten, *Boundary-Layer Theory*, 9th ed. Berlin, Heidelberg and s.l.: Springer Berlin Heidelberg, 2017.

[191] H. W. Liepmann and G. H. Fila, "Investigations of Effects of Surface Temperature and Single Roughness Elements on Boundary-Layer Transition," *NACA Annual Report*, vol. 33, pp. 587–598, 1947. [Online]. Available: https://authors.library.caltech.edu/448/

[192] G. C. Lauchle and G. B. Gurney, "Laminar boundary-layer transition on a heated underwater body," *Journal of Fluid Mechanics*, vol. 144, no. -1, p. 79, 1984.

[193] L. Lees, "The stability of the laminar boundary layer in a compressible fluid," *NACA Annual Report*, vol. 33, pp. 331–377, 1947.

[194] B. Loring, H. Karimabadi, and V. Rortershteyn, "A Screen Space GPGPU Surface LIC Algorithm for Distributed Memory Data Parallel Sort Last Rendering Infrastructures," in *Numerical Modeling of Space Plasma Flows ASTRONUM-2014*, ser. Astronomical Society of the Pacific Conference Series, N. V. Pogorelov, E. Audit, and G. P. Zank, Eds., vol. 498, Oct 2015, p. 231.

[195] T. Rinne, J. Hietaniemi, and S. Hostikka, *Experimental Validation of the FDS Simulations of Smoke and Toxic Gas Concentrations*, ser. VTT Working Papers, 2007, vol. 66. [Online]. Available: https://www.vtt.fi/inf/pdf/workingpapers/2007/W66.pdf

[196] A. M. Mathai and H. J. Haubold, *Linear Algebra: A Course for Physicists and Engineers*, ser. De Gruyter Textbook. Berlin, Boston: De Gruyter, 2017.